LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-SM-503118

ParaDyn, a Parallel Nonlinear Explicit , Three-Dimensional Finite-Element Code for Solid and Structural Mechanics: Version 11.1

A. J. De Groot, R. J. Sherwood, J. K. Durrenberger

Methods Development Group

September 30, 2011

Auspices

Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344.

Disclaimer

LLNL-SM-503118

ParaDyn User Manual

ParaDyn: A Parallel Nonlinear Explicit, Three-Dimensional Finite-Element Code for Solid and Structural Mechanics

Anthony J. De Groot, Robert J. Sherwood,
J. Kevin Durrenberger

Methods Development Group
Defense Technologies Engineering Division
Engineering Directorate

ParaDyn Version 11.1
September 2011

This page intentionally blank.

ABSTRACT

ParaDyn is a parallel version of the DYNA3D computer program, a three-dimensional explicit finite-element program for analyzing the dynamic response of solids and structures. The ParaDyn program has been used as a production tool for over a decade for analyzing problems which range in size from a few tens of thousands of elements to over ten million elements. ParaDyn runs on parallel computers provided by the Department of Energy's Advanced Simulation and Computing (ASC) Program at Lawrence Livermore National Laboratory and Los Alamos National Laboratory. In addition to these large-scale parallel computers, ParaDyn is now being successfully used on high-performance Linux clusters both at at LLNL and limited external sites.

Preprocessing and post-processing software utilities and tools are designed to facilitate the generation of partitioned domains for processors on a massively parallel computer and the visualization of both resultant data and boundary data generated in a parallel simulation. This manual provides a brief overview of the parallel implementation; describes techniques for running the ParaDyn program, tools and utilities; and provides examples of parallel simulations. The DYNA3D manual should be consulted for information on mechanics, discretization and algorithmic options.

This page intentionally blank.

PREFACE

Preface to Version 11.1

Updates in functionality are primarily documented by way of the new Release Notes in Chapter 5. Enhancements include the ability to use arbitrary node and element numbers without 1-to-*n* contiguous numbering and a more general 3-D discrete element formulation.

Preface to Version 10.1

Updates in functionality are primarily documented by way of the new Release Notes in Chapter 5. Enhancements include a first-generation implementation of a mortar contact algorithm leveraging past developments by Puso for implicit finite elements. Architectural revisions include final elimination of legacy memory allocation structures and having DynaPart utilize the DYNA3D input parser.

Preface to Version 9.1

Updates in functionality are primarily documented by way of the new Release Notes in Chapter 5. Several enhancements were made to ParaDyn as part of our collaboration with the Naval Surface Warfare Center, Indian Head Division. These support the next production release of the DYSMAS fluid-structure modeling system that will be the first to incorporate ParaDyn.

Preface to Version 8.1

Updates in functionality are primarily documented by way of the new Release Notes in Chapter 5. We also incorporated changes to Running ParaDyn Interactively on page 42, and added a new section about running coupled ParaDyn/GEMINI simulations on page 47.

Preface to Version 7.1

Updates in functionality are primarily documented by way of the new Release Notes in Chapter 5. We also incorporated changes to the command line syntax into Section 3.5 and Appendix 1 and marked with change bars.

Preface to Version 6.1

The ongoing functionality evolution captured for previous releases in this section is now collected as part of Section 5, Release Notes. We gratefully acknowledge the many collaborations with our users and colleagues that continue to contribute to the advancement of our capabilities.

Preface to Version 2.1/4.1

Message-passing versions of contact types 3, 5, 8, 9, and 10 are new in ParaDyn Version 4.1. These algorithms augment the local contact algorithms (types 1, 2, 3, 5, 6, 7, 8, 9, and 10) which place each contact interface into one processor. The new message-passing versions of local contact remove the limitation on processor count encountered when the largest local contact surface prevented further partitioning for more processors.

The significant milestone for Version 2.1 of ParaDyn is the release of highly optimized parallel automatic contact algorithms. These parallel algorithms incorporate dynamic load balancing and include penalty contact, Lagrange contact and SAND (material erosion) algorithms. The parallel algorithms in Versions 2.1 and 4.1 were designed for ParaDyn and DynaPart by Tony De Groot and developed by De Groot in ParaDyn and Bob Sherwood in the set of programs run by DynaPart.

The heightened interest and yearly assessments in Software Quality Assurance Processes and Practices have influenced our activities in this area. Software configuration management practices now include a split of the source code into two branches. The production version is characterized by carefully controlled changes to the source, limited to error corrections. The development version is the code-development source and incorporates new algorithms and modifications to existing capabilities. The development version plus alpha and beta testing then lead to a new production version. The version labels in this document refer to the production version of the ParaDyn software. The first two numbers in the version label represent the year of the release, measured from year 2000, and the number of the release within that year. Version 2.1 was released in 2002 and is the first (and only) release made in 2002.

Software Testing with defined alpha and beta testing periods was applied to Version 4.1, released in early 2004.

Analysts contribute significantly to the specification of new requirements, alpha/beta testing, and collaborating on the reporting and corrections of errors. We especially thank analysts in the Advanced Engineering Analysis Group at Lawrence Livermore National Laboratory (LLNL) and Ed Kokko at Livermore. Neil Hodge from the AEAG has provided the first beta testing of the VisIt parallel visualization tool. Thanks very much Neil! We also thank all of our collaborators, the Engineering Analysis Group at Los Alamos National Laboratory (Scott Doebling and Bob Stevens) and Photios Papados and Jim O'Daniel, at the Department of Defense Engineer Research and Development Center (ERDC).

Preface to Version 1.0

This User Manual documents the collaborative code development work by Carol Hoover, Tony De Groot and Bob Sherwood. It represent the majority of the original work in the parallel algorithm design and development for the ParaDyn Project. Doug Speck, Elsie Pierce, and Vic Castillo are now substantive contributors to the ParaDyn Project. Doug and Elsie, in particular, have made it possible to quickly design flexible binary databases (Mili) and have developed significant enhancements to the GRIZ visualization software. Their work paves the way for the visualization capabilities needed when using parallel computers with hundreds and thousands of processors.

Analysts provide the insights needed to turn our software development into an effective production tool for finite-element engineering analysis. At the Lawrence Livermore National Laboratory (LLNL) Dan Badders, Tony Lee, and Tony DePiero entered the turbulent waters of massively parallel computers earliest and have provided very valued input to our code development efforts.

Raju Namburu and Photios Papados are collaborators from the Army Research Laboratory (ARL) and the Engineer Research and Development Center (ERDC). They led the way to the first multimillion element calculations on Department of Defense high performance computers. John Benner and colleagues from the Los Alamos National Laboratory (LANL) were the first to use the ParaDyn program on over 1000 processors.

We especially appreciate the very valuable comments and support that our collaborators from ARL, ERDC, and LANL have provided for our code development efforts.

ACKNOWLEDGEMENT

The current ParaDyn team, and more broadly the entire Methods Development Group, is grateful for the years of leadership provided by Dr. Carol G. Hoover. Carol led the ParaDyn project from its research inception in the early 1990s through to successive production releases until the time of her retirement in 2005. These efforts included working with multiple internal and external sponsors as well as the subsequent transition to sustained investment by the ASCI program and its successor the Advanced Simulation & Computing program

We also wish to acknowledge to contributions of collaborative users from within LLNL and external sites that have provided feedback, asked probing questions, and thus helped mature the code in innumerable ways.

Table of Contents

1.0 BACKGROUND

Significant speed gains (for example, factors of a thousand on one thousand twenty-four processors) are being achieved for engineering design calculations on parallel computers using ParaDyn, the parallel version of the DYNA3D program [1]. The latest massively parallel computers with tens of thousands of processors now make it possible to design engineering models for mechanical system analysis with multiple component parts as well as to add more detail and complexity in the models for components. ParaDyn and DYNA3D are explicit finite-element programs designed to simulate the nonlinear, transient response of solids and structures. The two programs, contained within a single source, are developed by the Methods Development Group at the Lawrence Livermore National Laboratory (LLNL). The web site, http://www-eng.llnl.gov/mdg/mdg_home.html, provides general information and online documentation about the complete family of thermomechanical simulation tools developed in the Methods Development Group.

Parallel computers are now commonplace in our computing environment. Computers with speeds up to 10 PetaOps (10^{15} operations per second) and thousands of processors are been delivered to the Advanced Simulation and Computing (ASC) Program. These computers dominate the high-end computing at LLNL. Complementing these resources, the Laboratory has made a significant investment in distributed-memory Linux clusters. These systems are characterized by commodity processors (such as the Intel Xeon processors) with a high-performance interconnect network. The performance of ParaDyn on these systems depends very much on the balance between processor speeds and the speed of the interconnect network. A summary of the computer services and resources at LLNL may be viewed at http://computing.llnl.gov/.

The Department of Defense High Performance Computing and Modernization Program is also a very strong participant in acquiring hardware and developing software for next-generation massively parallel computers. Computer characteristics in terms of size, speed, and number of processors at the DOD Supercomputing Resource Centers (MSRCs) are summarized in the links provided by the Program Office web site at http://www.hpcmo.hpc.mil. These data illustrate the continuing significant increase in parallel computational speeds and capacities.

ParaDyn is a parallel production program. Code development efforts are now directed toward optimizing the parallel algorithms, developing parallel preprocessing and post-processing software, and developing software tools for engineering optimization studies. Concurrent with the parallel development effort, code developers for the DYNA3D program are contributing new

algorithms, elements, and material models to enhance the mechanics modeling capabilities. A single source encompassing both the ParaDyn and DYNA3D algorithms makes it possible to easily migrate these DYNA3D enhancements into the production ParaDyn program.

2.0 OVERVIEW OF PARADYN

2.1 Introduction

Advances in the development of parallel algorithms for explicit finite-element analysis and domain partitioning techniques have led to scalable production applications using ParaDyn. This has resulted in several benefits to our engineering design programs. Firstly, calculations are now performed in a day or less for problems that previously ran over several months. Secondly, new models are being generated for mesh sizes between one-million and ten-million elements. This is an order of magnitude larger than the largest models possible in the past. Finally, longer time simulations (problems running for a few million steps) are now being run on massively parallel computers.

Analysts have a challenging task in preparing models for parallel computers. The meshes are increasingly much larger and more complex. New validation tools are needed for the mesh generation step, specification of boundary loads and constraints, and defining facets on interfaces. In addition, the modeling of contact interfaces can have a significant effect on the parallel performance and scalability. Optimizing the performance and achieving scalability of parallel contact algorithms is particularly challenging and has been the focus of algorithm research in ParaDyn in the last several years. An overview of the parallel contact algorithms is presented in Section 2.4.

ParaDyn is a production program and includes a suite of software for automating the preparation of models and the analysis of results. The DynaPart preprocessing tool automates the task of model partitioning and is coupled directly to the research in scalable parallel contact algorithms developed in the ParaDyn program. The development of the GRIZ visualization tool [2] based on the Mili (Mesh I/O Library) software [3] provides a flexible, self-defining format to use for the large databases generated during a parallel simulation. Another recent advance which provides additional analysis tools has been the development of routines for reading families of Mili databases generated in parallel simulations. These routines have been implemented in VisIt, the parallel visualization tool developed by the ASC Program at Livermore and also in EnSight, a commercial parallel visualization tool supported by the ASC Program and used at Los Alamos.

The ParaDyn software suite consists of the following software products:

- DynaPart, a partitioning tool for the spatial decomposition of the model input data;
- DYNA3D, a nonlinear, explicit, three-dimensional finite-element code for modeling the dynamics of solids and structures;
- ParaDyn, the distributed memory parallel version of DYNA3D;
- Mili, an Input/Output (I/O) subroutine library for formatting binary databases used in finite-element computer programs;
- Xmilics, a tool for combining families of Mili databases created on a parallel computer;
- GRIZ, a visualization tool for post-processing results.

There are three additional visualization tools that have been used for evaluation of results in Mili databases from ParaDyn. These are:

- VisIt, a parallel visualization tool for post-processing results.
- EnSight, a commercial parallel visualization tool used by analysts at Los Alamos.
- DYSMAS/P, a visualization tool for post-processing results.

For installations where the ParaDyn software suite is available, all of the documentation is in PDF form and stored in directories with the executables and related libraries. Other individuals or groups interested in the capabilities of the software developed by the Methods Development Group (MDG) can access documentation on the MDG home page. The home page web address is http://www-eng.llnl.gov/mdg/mdg_home.html. The VisIt software includes PDF documentation and instructions for downloading the software. The home page for VisIt is http://www.llnl.gov/visit.

To prepare input data and select control options and flags for the ParaDyn program, follow the discussions in the DYNA3D User Manual [1]. Use Chapter 3 in this manual to follow the steps needed to partition a model, run the problem on a parallel computer, and use the post-processing tools for analyzing the results. Instructions for using the DynaPart preprocessing software are outlined in Section 3.3 and discussed in depth in Appendices 1-3. The Xmilics tool is discussed in Chapter 3 and a Help package is displayed interactively by using the execute line with no arguments. The Xmilics user manual is included in Appendix 4. The post-processing software is documented in the GRIZ User Manual. Finally, some standard features in DYNA3D requiring additional documentation for a parallel analysis are included in Chapter 4 of this manual.

A set of typeface conventions is followed throughout this manual to allow the reader to easily distinguish between **commands**, *parameters*, and `computer generated text`. **Commands** that appear in **bold** type should be entered verbatim. *Parameters* that appear in *italic* type should

be given values when included in the input. `Computer generated text`, such as error messages or default file names, is printed in a `typewriter-like (Courier)` font. In text passages file names appear in italic type for clarity.

The next sections provide introductory discussions about parallel algorithms and computers. Section 2.2 discusses the parallel finite-element model and describes partitioning methods. Section 2.3 discusses parallel performance and scalability measurements. Section 2.4 characterizes contact interfaces and their implementation in parallel. Section 2.5 lists boundary conditions and other options which are given special treatment by the partitioning software. Section 2.6 discusses scalability studies for different models and parallel computers.

2.2 The Parallel Finite-Element Model

A successful strategy for parallel implementation of the explicit finite-element method is based on dividing the mesh among the processors and executing ParaDyn on a subdomain in each processor [5]. The elements from the mesh are divided into subdomains so that each processor has approximately the same amount of calculations to perform in a timestep. The nodes on the boundaries of a subdomain are referred to as shared nodes. Nodal force data for shared nodes are communicated between processors when the nodal force updates are calculated. The nodal points on the subdomain boundaries are duplicated (shared) in more than one processor. Mesh partitioning is the strategy for dividing the problem into subdomains and mapping subdomains to processors. This is illustrated for two processors in Figure 1.

The nodal forces consist of contributions from applied loads, contact interactions, and internal deformations.

$$\mathbf{F}_{\text{node}} = \mathbf{F}_{\text{applied}} + \mathbf{F}_{\text{contact}} - \mathbf{F}_{\text{internal}}$$

The internal force calculation for a shared node includes a contribution from elements in different processors. Each processor calculates a partial nodal force for the elements in its processor. These partial force contributions are communicated among the processors so that the total force computed for a shared node is the same in all processors. As much as possible, the ParaDyn algorithms are designed to store partial nodal force data for shared nodes until all contributions to the nodal force have been computed before communicating the shared data.

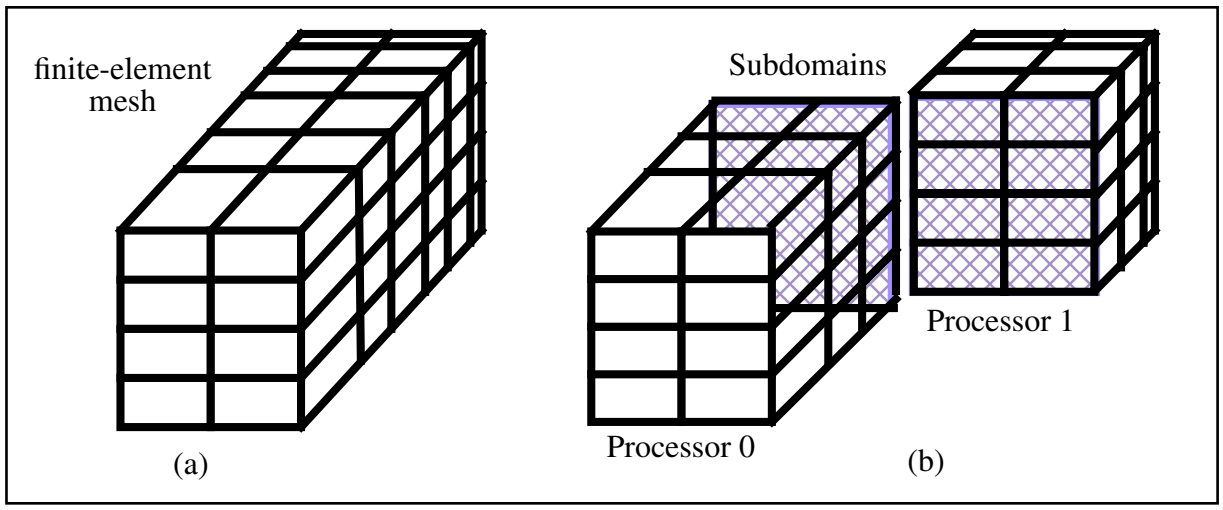


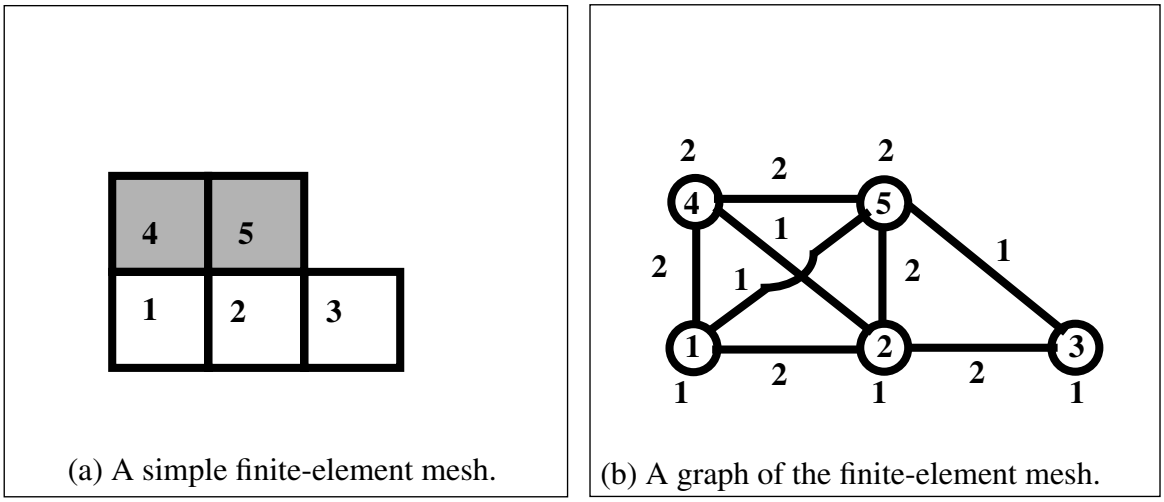
Figure 1. Two subdomains on a finite-element mesh. (a) The original mesh with 48 elements is partitioned into two subdomains with 24 elements each. (b) The calculations involving nodal points on the cut plane (shown as patterned) are performed in both processors. The 15 nodes on the cut plane are referred to as shared nodes.

Research in applied mathematics has led to efficient techniques for subdividing or partitioning the complicated unstructured meshes that arise in practical engineering applications [6-9]. We use the Metis software from the University of Minnesota to partition finite-element meshes and contact surfaces. (For more information on Metis, see http://www-users.cs.umn.edu/~karypis/metis/main.shtml). The Metis algorithms use a graph to represent the finite-element mesh. Preprocessing software automatically produces the graphs needed for the mesh partitioning step.

Mesh partitioning is accomplished by representing a finite-element mesh as a graph. A graph has vertices and interconnecting edges. The vertices and edges represent objects on the mesh. For finite-element meshes, the vertices of the graph correspond to elements (zones) in the mesh and the edges correspond to nodes in common between two connected elements. See Figure 2.

The graph represents the element-to-element connectivity for the mesh. The Metis algorithms find an efficient division of the graph corresponding to a specified number of subdomains. An important aspect of graph partitioning techniques is the use of weighting factors for both the vertices and edges. These weighting factors are used to balance the vertices into sets of roughly equal weight. This weighting of the graph provides the best representation of both the computational costs and the communication costs for the partitioning of the mesh. To illustrate this, the relative

computational cost for a complex material model, a boundary condition or any other expensive part of the calculation, can be used to weight the vertices. Similarly, an edge in the graph represents the number of common nodes between the elements and can be appropriately weighted by a relative measure of the shared data communicated if the graph is cut on that edge. Figure 2.(b) indicates the edge weights used for the few elements shown in Figure 2.(a).



(a) A simple finite-element mesh.

(b) A graph of the finite-element mesh.

Figure 2. A finite-element mesh and the graph representation of the mesh. (a) This simple mesh consists of two materials, shaded and unshaded. The shaded material requires twice as much calculation time as the unshaded material. (b) This is the graph of the mesh. The vertices are represented as circles and the number near a vertex is the computational weight of the vertex. The lines connecting the vertices are edges and represent the shared data between the vertices. The number specified along the edges represents the number of shared nodes between the two elements represented by the vertices.

The partitioning task is automated completely in the preprocessing tool, DynaPart, for any mesh geometry. This software was used for the assignment of a one-million element mesh to 128 processors as shown in Figure 3. The colors are used to show the processor assignment for subdomains on the mesh. The mesh was developed to model a shock moving from the top vertex of the mesh in the direction of the half-cylindrical cavity region located half way down and on the left-hand side of the mesh. The mesh is zoned very finely at the top of the model to resolve the shock structure. As a result of this fine zoning the subdomains are much smaller at the top of the mesh than the subdomains on the lower part of the mesh where the zoning is coarse. The mesh lines are not shown in Figure 3.

.

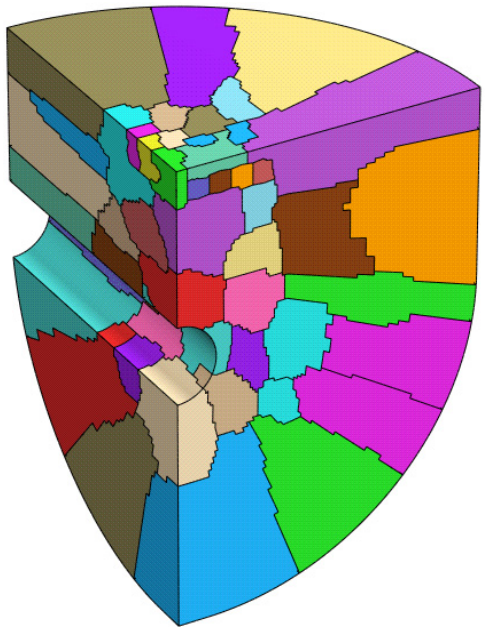


Figure 3. Processor assignment for a one-million element mesh. Colors are used to distinguish the subdomains assigned to 128 processors. This problem without sliding interfaces scales linearly as the number of processors is increased up to roughly 1000 processors.

2.3 Parallel Performance and Scalability

The time required to deliver results on a parallel computer is the sum of the time for computing results on the processors and the time for communicating data between the processors.

$$\tau_{wc} = \tau_{calc} + \tau_{comm}$$

where

τ_{wc} is the total elapsed time taken for the calculation (total wall clock time);
τ_{calc} is the elapsed time the processors spend computing results;
τ_{comm} is the elapsed time the processors spend communicating shared data.

Ideally, if the number of processors is doubled, the rate for delivering results will be doubled. The term *scalability* or *theoretical scalability* refers to this linear scaling of the delivery rate with the number of processors. In practice scalability breaks down when the communication time becomes

significant compared to the amount of time processors spend computing results. Speedup curves measure the calculation rate versus the processor count for specific computers. Examples are shown in Figure 6.

If the parallel calculation is efficient, the delivery time τ_{wc}, for N_p processors will be approximately equal to $1/N_p$ of the time for calculating the same results on one processor, τ_1. Thus, a measure of the parallel efficiency is given by the following:

$$\varepsilon = \tau_1/(N_p\tau_{wc}) \times 100\%.$$

Almost all problems of interest include contact interface definitions which are often challenging to run efficiently in parallel. See Section 2.4. One of the software programs in DynaPart (reducegrf) calculates an estimate for the maximum number of processors that can be used for a ParaDyn simulation based solely on balancing the computational work for *both* the element deformation and the contact calculations. Note that interprocessor communication time is ignored in this estimate. Using more processors than this estimate will result in no further improvement in delivering results and wastes computing resources. On the other hand, an efficient calculation may be limited by the number of processors even further for any specific hardware platform. In this case, hardware speeds, CPU and network are taken into account. Section 2.6 illustrates this with examples.

2.4 Scalable Parallel Contact Algorithms

Designing efficient parallel contact algorithms is challenging for several reasons:

- A load balanced mesh partition may split a contact interface into many processors and potentially cause a large amount of unnecessary communication of contact nodes and facets.
- Communication may be necessary for parallel contact algorithms if the relative distance moved by the two surfaces is more than one element length.
- For problems with moving parts, the material interfaces in contact change dynamically. Search algorithms for finding interfaces in contact are relatively time consuming compared to the contact enforcement or element deformation calculations.

Because of these characteristics of the interface calculations, dividing the problem domain into subdomains that are optimal for calculating the element deformations may not result in an efficient division of the problem for the contact calculations. In most cases the ParaDyn software uses partitioning methods for the sliding interfaces that differ from the partitioning for the mesh. As a result the contact calculations are performed in a different set of processors than the element calculations for connected elements. The contact force results are communicated once in a timestep to the processors defined by the mesh partition. Similarly the nodal position and velocity updates are communicated from the processors defined by the mesh partition to the processors defined by the contact partitions once in a timestep.

There are currently two classes of parallel contact algorithms in the ParaDyn program: *local contact* and *arbitrary contact*. In some problems, surfaces initially close together engage in small relative motion and the contact remains in a localized region of the mesh. We refer to this as *local contact*. Sliding interface types (1-3, 5-10) in DYNA3D/ParaDyn are *local contact* algorithms. For other problems with many components and with large relative motion at the interfaces, the interactions of the surfaces are not predictable. Thus, more sophisticated and time consuming searches for the surfaces in contact must be performed throughout the simulation. We refer to this as *arbitrary contact.* Examples illustrating arbitrary contact are a ball rolling on a plane, a surface folding on itself, material fragmentation and failure, or an automobile crash simulation. Arbitrary contact is implemented in DYNA3D automatic contact interfaces, types 12 through 14. The automatic contact algorithm type 14 is the material erosion algorithm (SAND).

The parallel arbitrary (automatic) contact algorithm uses a localization technique (a bucket search) to search for contact points. The partitioning of arbitrary contact surfaces is a step separate from the mesh partitioning. Thus, there are two distinct partitions of the model and there is communication between the two partitions at each timestep. The parallel arbitrary contact algorithm also redistributes the contact surfaces over the processors if the surface motion results in motion over a distance larger than one bucket length. This redistribution step is referred to as *dynamic load balancing.*

Sliding interface type 11 (SAND) has been removed because the equivalent interface is modeled in the more robust and newer automatic contact interface type 14.

The most commonly used sliding interface in DYNA3D is type 3. The same physical interface condition (sliding with friction and voids) can also be modeled with the automatic contact algorithm types 12 and 13. The parallel efficiency for either choice for the sliding interface may vary significantly depending on the details of the interfaces and the size of the model. The next sections discuss the two forms of parallel contact algorithms.

2.4.1 Parallel Local Contact

Starting with ParaDyn Version 6.1, DynaPart divides local contact interface types 3, 5, 8, 9, and 10, if the largest contact surfaces limit the number of processors that can be used. For more than one local contact sliding interface, DynaPart distributes the set of interfaces among as many processors as possible. Special graph weighting methods are used for partitioning the local contact interfaces and evenly distributing the local contact calculations among processors.

Figure 4 shows a model of Morrow Point Dam, a dam located in the Black Canyon of the Gunnison River in Southwestern Colorado. This model contains 1,071,625 solid hexahedral elements and 1,120,280 nodes. The model uses about 20 type 3 contact surfaces. The simulation performed on this structure used ParaDyn for both the gravity and seismic portions of the analysis on the MCR higher-performance Linux cluster formerly at Lawrence Livermore National Laboratory. The dam was analyzed for a M 6.5 earthquake on the Cimarron Fault. The finite element model not only includes a topographically accurate flexible foundation, but also considers nonlinearities due to the shear keys across the vertical contraction joints, nonlinearities due to the extensive use of contact surfaces in the dam/reservoir/foundation system, and material nonlinearities due to concrete cracking.

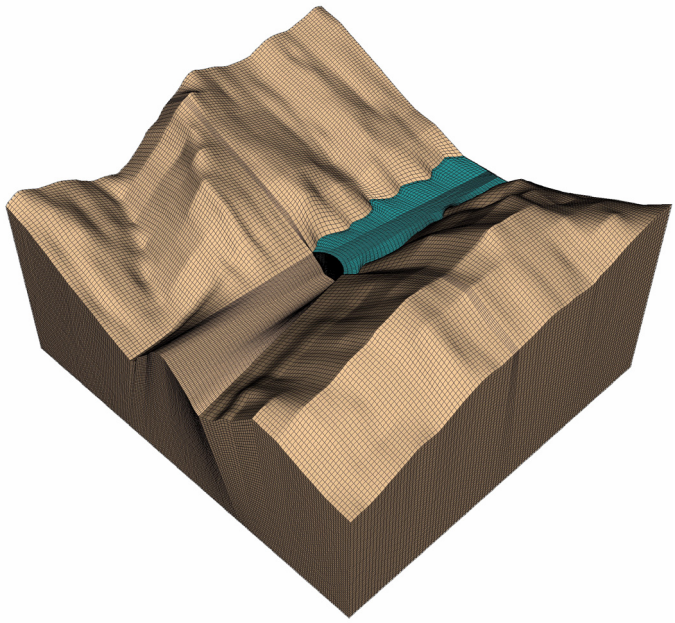


Figure 4. Million element model of Morrow Point Dam. It uses twenty Type 3 contact surfaces. Courtesy of Chad Noble.

2.4.2 Parallel Automatic Contact

For arbitrary contact defined with sliding interface types 12-14, the analyst avoids the very time consuming task of defining the contact surfaces in the model. However, the parallel algorithm for these interface definitions is more expensive for two reasons. First, the search for contact is an expensive step, both on single processor computers and on parallel computers. Furthermore, for problems with considerable surface motion, the parallel versions of these algorithms require additional data communication because the buckets on processor boundaries can be located in multiple processors. A periodic surface rebucketing is needed (on serial and parallel computers) to avoid missing contact points.

An example of arbitrary contact is shown in Figure 5. This is a hypervelocity impact problem. Type 14 contact is used, with the free node option.

The first step in the arbitrary contact algorithm is a sort step to localize the search for material interfaces. The sort algorithm generates a set of buckets (cells) for grouping surface nodes and facets into localized regions on the mesh. The parallel version of the algorithm partitions the

buckets among the processors to balance the contact calculations among them and minimize the communication of nodes and surface facets in buckets with data that must be shared between processors.

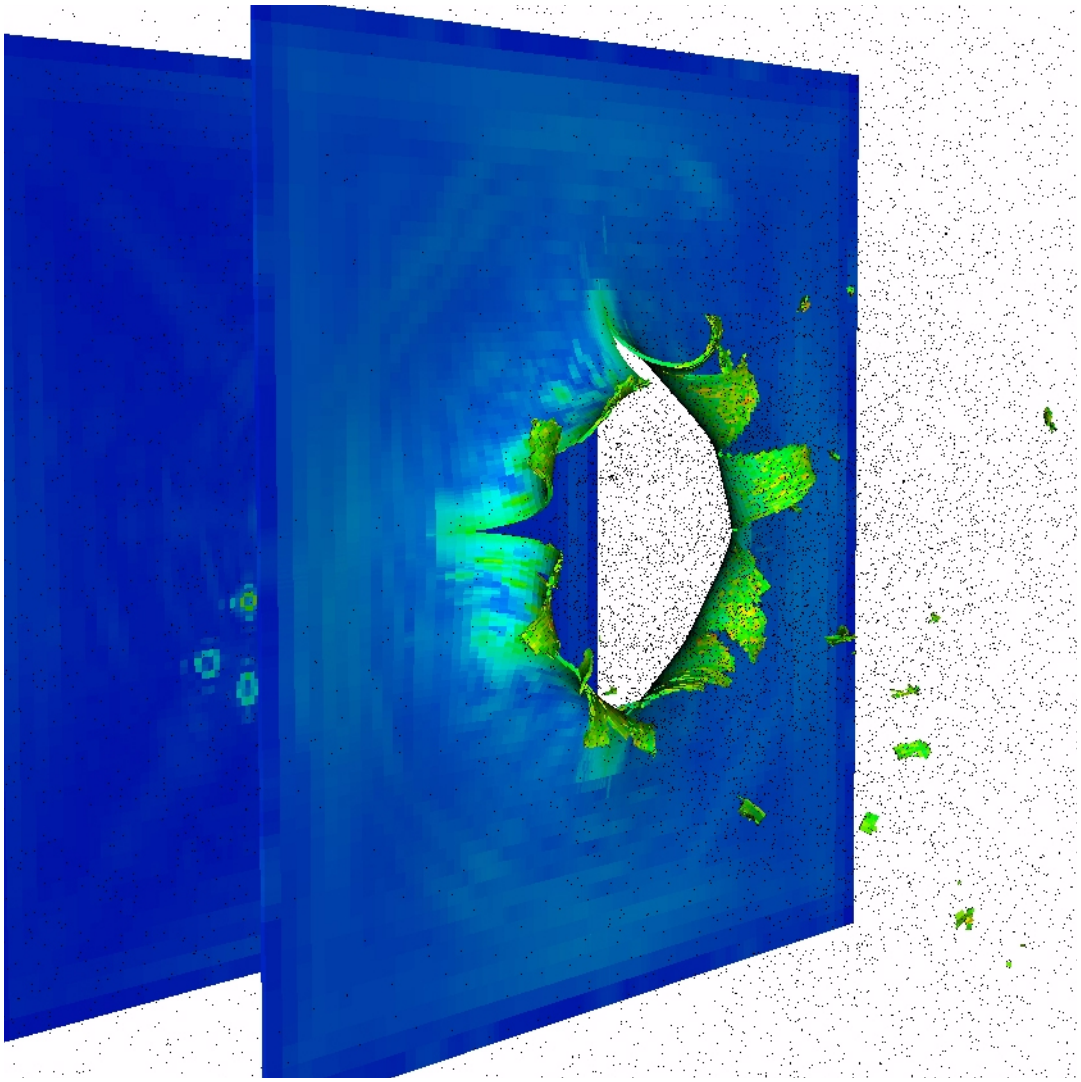


Figure 5. Simulation of a hypervelocity impact through two plates utilizing the material failure model of Material 64 and the free node capability to convect momentum associated with failed elements. Figure courtesy of Doug Faux.

An additional step is needed in the automatic contact algorithm when the surface motion results in nodes and facets moving more than one bucket width. When this happens it is necessary to resort the surfaces so that contact points are not missed. The frequency of this bucket-regeneration step

is automatically computed. In ParaDyn, the bucket-regeneration step also includes a partitioning of the buckets to load balance the new set of buckets among the processors. The bucket regeneration and partitioning require extra communication during the timestep over which it occurs. Thus, it is important to rebucket as infrequently as possible to avoid the extra communication costs.

The efficiency of the parallel arbitrary contact can be improved by limiting the search domains with the DYNA3D keyword input options listed in the section for contact algorithms. The options for controlling the search domain and other features are as follows:

- Boxes can be defined to limit the domain of the search;
- Material can be included or excluded in the boxes for the search;
- Faces defined as in a type 3 interface can be specified rather than automatically generated.

Refer to Section 4.1 for a discussion of modeling tips for developing models that result in the most efficient performance from the parallel contact algorithms.

2.4.3 General Guidance for Efficient Parallel Contact

In order to maximize computational performance, all contact surfaces should be kept as small as possible. If any contact surface can be defined as two or more independent smaller surfaces, this should be done. For efficient parallel arbitrary contact (automatic contact), one can use DYNA3D input options to limit the search domains. For a complex mesh it is possible to use both local and arbitrary contact algorithms and to provide multiple instances for each type of contact. An obvious advantage in doing this is that the regions on a mesh without any contact are not included in either the partitioning for local contact or the search domains for arbitrary contact.

Section 4.1 provides valuable *modeling tips* for all forms of parallel contact. It is very important to read Section 4.1 before designing the contact for models that will be run using ParaDyn.

2.5 Boundary Conditions and Constraints

Parallel versions of boundary conditions and constraints are treated both in the partitioning software as well as in ParaDyn. Because the partitioning software uses special processing on selected boundary conditions and constraints, it is important to know which boundary conditions and constraints are treated with partitioning and how this affects the overall partitioning.

The following DYNA3D options (and DYNA3D User Manual section number) affect the partitioning by placing all associated nodes and elements into one processor. The largest set of nodes for any of these features will limit the maximum number of processors that can be used efficiently. It is therefore best to keep the size of each set of these boundary conditions and constraints as small as possible.

Symmetry planes with failure (4.15)
Follower forces (4.22)
Nodal constraints (4.26)
Sliding interface types 1 or 4 (4.28)
Tie-breaking shell slidelines (4.29)
Tied node sets with failure (4.30)
Rigid body joints (4.34)
Shell-solid interfaces (4.39)
One-dimensional slidelines (4.45).

These objects contain nodes and elements that must be assigned to a single processor rather than divided across more than one processor. Nodes that need to be kept together are assigned to Special Nodal Points (SNP) sets in the partitioning software. Associated with each of the SNP sets is a Special Element (SE) set. The SE set consists of all elements that contain one or more nodes in the corresponding SNP set. Each SNP set and its associated SE set must be fully contained in a processor. These current ParaDyn limitations could be relaxed if sufficient interest from specific applications warranted the development investment.

2.6 Testing and Evaluating Model Scalability

If processors are readily available on the parallel computer, it is a good idea to partition the model more than once in order to select an optimal number of processors to use for the simulation. Selecting an optimal number of processors means you use as many processors as you can (and thus,

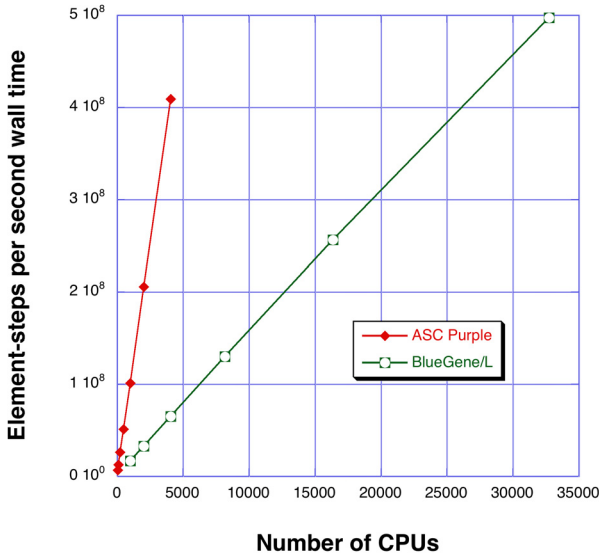
get the best turnaround for your results) and at the same time you run the parallel simulation efficiently. The partitioning software provides statistics that allow you to make a reasonable estimate for the number of processors to use for achieving a computational load balance. On the other hand, it may not be possible at the partitioning step to show that the communication time is negligible compared to the calculation time.

Therefore, it is necessary to benchmark the ParaDyn performance with scalability studies for all parallel machines that will be used and for the prototype models of interest. This step is not necessary if other analysts have done this study already.

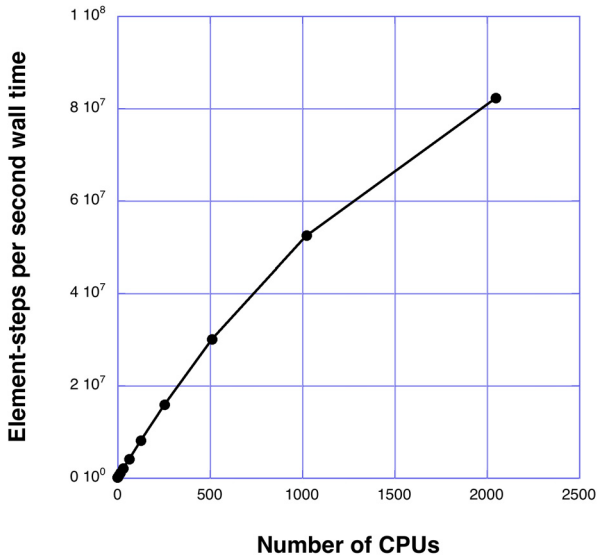
Tests using the current systems at LLNL have shown that processing speed increases with the number of processors with reasonable efficiency until the number of elements per processor drops below approximately 2000 to 4000. This estimate may be inadequate if contact surfaces, boundary conditions, or nodal constraints are restricting the partitioning. The reason is that the optimization of the parallel algorithms for these options constrains the nodal data to reside on one or a small number of processors. See discussions in Sections 2.4 and 2.5. Finally, it is important to generate enough points on the speedup curve to determine the number of processors at which the speedup curve begins to deviate significantly from the ideal linear speed up.

2.6.1 Speedup Studies and Scalability Limit

Speedup studies and the scalability limit are important to understand for any computer platform on which ParaDyn is used. To generate a speedup curve, the wall clock time is measured as a function of the number of processors. Then the rate of calculation is plotted as a function of the number of processors. Perfect scalability or *theoretical scalability* means that the calculation rate is doubled as the number of processors is doubled. Thus a perfectly scalable calculation is one with a straight line at 45 degrees when the calculation rate is plotted as a function of processor count. In practical problems a straight line behavior with a slope less than the theoretical scalability is sufficient and usually indicates that communication time is affecting the rate at which processors receive data needed to complete calculations for nodal data shared between processors.



a) Parallel speedup of a simple, homogeneous 90 million-element problem.



b) Parallel speedup of a one million-element model with a large contact surface.

Figure 6. a) Speedup for a 90 million-element problem having no slidesurfaces defined. A small percentage of the nodes are communicating between processors. Note that performance is still increasing at more than 32,000 processors. b) Speedup curve for a one million-element problem having one large Type 3 slidesurface. Although performance continues to increase, the rate of speedup is beginning to decrease at approximately 1024 processors. For the 1024-processor case, there are only about 1000 elements per processor. Such small computational work per processor lowers parallel efficiency.

In ParaDyn, timing values are printed in the standard output file (d3hsp) each time the problem energy statistics are printed. These statistics are useful because they provide an indication of the extra cost of an automatic contact rebucketing timestep compared to a step without rebucketing. For the speedup curves the final timing statistics measured for the problem and printed at the end of the standard output file should be used to compute the calculation rate. This value at the end of the run amortizes the differences in the time for timesteps with and without rebucketing for contact. Speedup curves for problems with and without contact are illustrated in Figure 6.

The processor count at which there is no significant deviation from linearity in the speedup curve is usually selected as the maximum number of processors for which the problem is reasonably scalable. For ParaDyn beyond this point on the speedup curve may be caused by large and unpredictable communication times. This point in the speedup curve may be due to a load imbalance, caused by the features described in section 2.5. DynaPart provides enough information that can be used to avoid the load imbalance caused by these features. DynaPart programs collect statistics about the model objects that require splitting to maintain good load balance and the statistics on the cut edges for each partition. From these data DynaPart estimates and prints the processor count at the scalability limit and labels it “Max # partitions” (See discussion in Section 2.6.2). This processor count is an estimate of the scalability limit to load balance the calculations. It does not account for the hardware communication times and the balance between the hardware CPU and interconnect communication times. It is for this reason that the speedup curves should be studied for each platform on which ParaDyn is used.

2.6.2 Using Statistics from the Partitioning Software

After gaining an understanding of the performance characteristics of ParaDyn for specific parallel platforms and prototypical models, it is possible to use statistics in the DynaPart log file as a guide for selecting the number of processors.

Computational Load Balance:

The **skmetis** program in DynaPart calculates and prints statistics for evaluating the quality of the partitioning with respect to the computational load balance among the processors. This output is listed as the *Load balance* number from Metis. The best balance is obtained when this number is as close as possible to a value of 1.00. The ideal *parallel speedup* is given by the ratio of the number of processors to the load balance. This assumes the communication time is negligible and the deviation of the speedup from the ideal is due to the imbalance in the calculations across processors.

The UNIX **grep** command shown below prints the line in the DynaPart log file containing the computational load balance statistic.

grep "Load balance:" *logfilename*

```
Load balance:          1.010997      Edge cuts:              427586
```

Maximum Number of Processors Limited by Features Described in Section 2.5:

The partitioning of models that have boundary constraints listed in section 2.5 may limit the number of processors that can be used for an efficient ParaDyn simulation. The DynaPart software computes and prints a good estimate for the optimal number of processors to achieve computational balance when these options are used in the model. If this estimate is less than the requested number of processors, the model can then be repartitioned using the *minimum* of this value and the optimal the number of processors found on the speedup curve. Use **grep** to get this statistic from the DynaPart log file as follows:

grep "Max # partitions" *logfilename*

```
Max # partitions for good load balance:       45
```

Balancing and Limiting Communication Time:

The load balancing of the computational work by Metis provides a very good estimate for the achieved speedup with ParaDyn for models that do not use contact. However, it is not possible to develop an equivalent estimator for the effect of the communication time on the scalability without running speedup tests with your model on the parallel computer you are using. It is particularly important to do this for models that use contact.

This page intentionally blank.

3.0 ANALYSIS WITH PARADYN

An important consideration in developing a parallel simulation capability is the requirement to provide automated tools for partitioning complex finite-element meshes and for managing the hundreds of files generated by a parallel simulation. The ParaDyn software suite includes additional tools for these tasks. This section describes the ParaDyn software products and how to use them.

3.1 The ParaDyn Software Suite

The ParaDyn program the implementation of the DYNA3D program for parallel computers. The input file for ParaDyn is the same as the DYNA3D input file and can be prepared using the DYNA3D manual and a mesh generator in the usual way. The term *model* used here includes the mesh, all of the initial and boundary conditions, and the selected mechanics options; in other words, all of the data in the input file. The steps for preparing and running a ParaDyn simulation are as follows:

Step 1. Mesh generation and model preparation.
Prepare your input using a mesh generator and the DYNA3D manual.

Step 2. Model Partitioning.
Divide the DYNA3D model into subdomains, one per processor.
Determine the optimal number of processors to use.

Step 3. ParaDyn Analysis.
Run the ParaDyn analysis.

Step 4. Combine the binary output databases for visualization (optional step).
Combine the families of databases from each processor (subdomain) into a family of files in a single database (full domain).

Step 5. Visualization.
Display results on the parallel computer or display results on a workstation.

Step 1 is the usual data preparation step for the problem. Step 2 is a preprocessing step for model partitioning and this step is automated with a script file, DynaPart. The DynaPart script runs several programs for dividing the DYNA3D model into subdomains and creating lists of nodes that share results between processors. In Step 3 ParaDyn runs DYNA3D on each of the processors using the subdomain of the mesh assigned to that processor and, when needed, exchanging data among

processors during each timestep of the calculation. Each processor generates a family of text and binary files. Step 4 is used only when post-processing with GRIZ. The Xmilics utility is used in Step 4 to combine the binary database family for each subdomain into a binary database family representing results on the full mesh. In Step 5 there are four software products that can be used for visualization: GRIZ, VisIt, EnSight, and DYSMAS/P. VisIt and EnSight read Mili databases and render images in parallel. DYSMAS/P is a visualization tool that can read mili databases and the output from DYSMAS. EnSight is used at LANL and VisIt is available for use with ParaDyn at LLNL.

The VisIt visualization tool can be used to visualize the parallel databases *without* the combine step. It can be used on a parallel computer or in a client/server mode with the parallel computer as a server (for parallel rendering and computing) and a workstation as a client (for displaying images, for fast interactivity, and for processing mouse commands). VisIt is a free interactive parallel visualization and graphical analysis tool for viewing scientific data on Unix and PC platforms. For more information on VisIt, the website is http://www.llnl.gov/visit/about.html

The software products used for ParaDyn simulations are summarized in Table 1. This table includes the input needed and output generated for each software product.

The next sections describes how to use these products with example execution lines. The final section summarizes the steps for using the ParaDyn software and includes an execute line for each product. This summary can be used as a handy reference once the details of each step are understood.

Table 1. ParaDyn Software Products.

Task	Software	Input	Output
Partition the model.	dynapart	DYNA3D input file.	Partition file; Plot file for the partitioned mesh.
Run a parallel simulation.	paradyn	DYNA3D input file, Partition file.	Families of files from each processor include: Plot and time history output; Restart dump files; Text output files.

Table 1. ParaDyn Software Products.

Task	Software	Input	Output
Combine the binary databases.	xmilics	State and time history databases for each processor (subdomain); partition file	State and time history databases for the full mesh.
Visualize the results.	griz4s	State and time history databases for the full mesh. State databases for each processor; partition file.	Screen display; RGB, postscript, PNG and JPEG output files.
	visit	State databases	Client/server screen output, movies, and other graphics formatted output files.
	ensight		
	DYSMAS/P	State databases	

3.2 PATH Variable for Accessing ParaDyn Software

The directory containing ParaDyn software should be included in the path names in the UNIX PATH variable. On all computers at Livermore Computing, the following **set path** command should be added to the .cshrc file to point to the ParaDyn software products.

set path = (/usr/apps/mdg/bin $path)

The documentation is located in the directory */usr/apps/mdg/doc* and */usr/gapps/mdg/doc* on all computers at Livermore Computing. On the Engineering servers at LLNL, the documentation is located in the directories */usr/gapps/mdg/manuals* and */usr/gapps/mdg/doc* for the unclassified and classified networks, respectively.

3.3 Partitioning a Model

The DynaPart script is used for *partitioning* the model input. The model includes the mesh, all of the initial and boundary conditions, and the selected mechanics options; in effect the standard DYNA3D input file. The partitioning subdivides the mesh, contact surfaces, constraints, and other boundary conditions that share nodal data when the domain is divided among the processors. See Chapter 2 for details.

Running DynaPart in a subdirectory is recommended. This is because the partitioning software generates a number of intermediate files in the current working directory. If these files from the first partitioning of a model are retained by using keyword *keepfiles* and are not modified, then DynaPart with the keyword *again* can be used for subsequent partitioning of the same model for a different number of processors if no other keywords are changed. DynaPart uses results saved from the original DynaPart run and consequently can skip over some of the time consuming programs executed by the DynaPart script. It is advisable to use the *again* keyword on the DynaPart execute line when repartitioning a large model for a different number of processors. Finally, using a subdirectory also conveniently groups the files containing the results generated for each model that is partitioned.

The form of the DynaPart execute line is

dynapart *infile np [keyword1 keyword2 ...]* **|& tee** *logfilename*

The first two arguments are required. The first argument is the name of the ParaDyn input file and the second argument is the number of processors requested for the partitioning. The remaining arguments are optional keywords. These keywords are described in detail in Appendix 1. For most models, using the dynapart command without any keywords will result in a high quality partition. The most commonly used keywords are *keepfiles* and *again.* The keyword *keepfiles* is used to cause DynaPart to not remove intermediate files at the end of a run. The keyword *again* is described above. The output from DynaPart is piped into the UNIX **tee** utility. This step should always be used so that the statistics generated by the partitioning programs as well as other diagnostic messages can be examined after DynaPart is run.

The following is an example of the DynaPart execute lines for the input file *infile* and for partitions for 64 and 128 processors. Notice that the first execution line uses keyword *keepfiles* so that the second execution line for the same model can use keyword *again* to save computer time.

dynapart infile 64 keepfiles |& tee infile64.log First partition of a model for 64 processors

dynapart infile 128 again keepfiles |& tee infile128.log Repartition for 128 processors

The output from the mesh partitioning is a file that is referred to as the *partition* file. The partition file is a required input file for the ParaDyn program.The name of the partition file is *infile.np* where *np* is the number of processors selected. For the *d3samp1* input file, the name for a four-processor

partition is *d3samp1.4*. The first several lines in the partition file contain processor (subdomain) totals for the nodes and elements assigned to each processor. This file can be viewed with a text editor.

DynaPart generates a Mili plot database (binary result files) that can be used to visualize the subdomains of the partitioned model. The root name for the plot file is *parplt*. In this special case, there is only one file in the Mili database and this file contains only the geometry for the model. The last character in the geometry file name is always an *A and if you change to a new file name, the last character must also be an A*. GRIZ can be used to visualize the database. The result is a coloring of the mesh subdomains by processor and a corresponding map of color values associated with the processors. The color map labeled *materials* is a coloring by processor for plots generated by DynaPart.

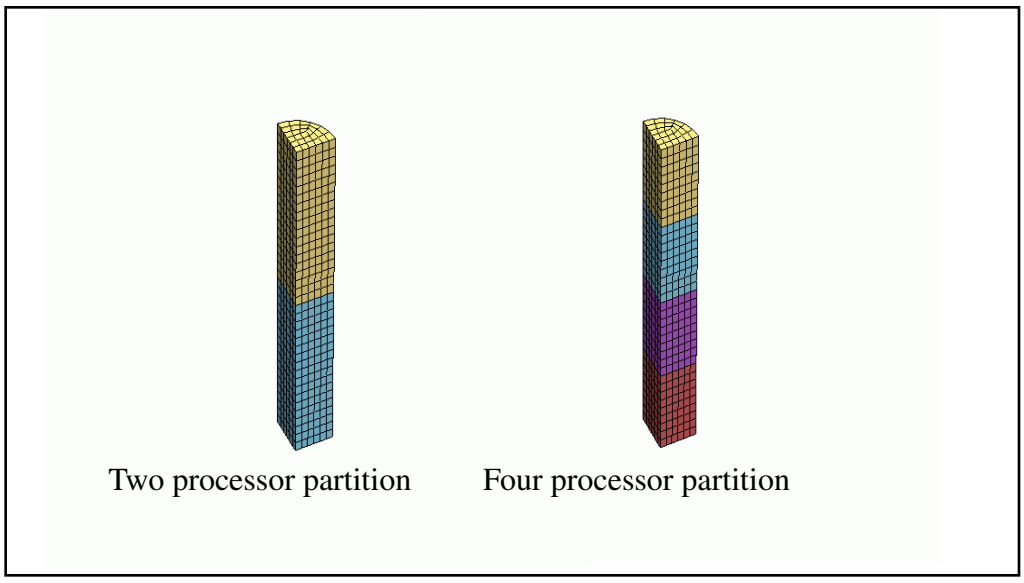


Figure 7. Two and four processor partitions for the *d3samp1* test problem.

Caution: The database file, parpltA, is overwritten when DynaPart is run again in the same directory. The database can be saved by renaming it before partitioning the model another time. Remember the last character in the new name must be an *A*.

There are two important statistics written to the DynaPart log file, (1) the load balance computed in the Metis programs and (2) if local contact or other special options are used, a processor count is printed which corresponds to the maximum number of processors that can be specified to achieve a computational load balance. The optimal load balance number is 1.000 and generally the load

balance should be less than 1.1 (a 10% load imbalance). See discussions in Section 2.6. These two numbers can be easily viewed by using the following UNIX **grep** commands on the DynaPart log file as shown in the following examples:

grep -i "load balance:" *logfilename*

```
Max # partitions for good load balance:       27
Edge cuts:    1080,  Load balance:  1.00000,  Score: 1.55556.
```

Example 1. Mesh partitioning and repartitioning for a simple model with no contact
This very simple example illustrates the use of DynaPart for model partitioning. This problem does not include the special options, such as contact or nodal constraint sets, that constrain the partitioning of the mesh. Thus, there will not be a statistic computed by DynaPart for the limit on the maximum number of processors, and the load balance should be very close to 1.000.

Step 1. Set up a subdirectory for partitioning the *d3samp1* problem

mkdir partition	Make a subdirectory *partition*
cd partition	
ln -s ../d3samp1 .	Make a link to the input file, *d3samp1*

The UNIX utility **ln** links the *d3samp1* file to the subdirectory *partition* without making a copy. This saves disk space when working with large files. For the same reason it is a good idea to use **ln -s** in subdirectories when making runs using the same input file and varying the number of processors. This is the case, for example, when generating the speedup curves for a large model.

Step 2. Partition the model for two processors and view the partitioned results with GRIZ.

dynapart d3samp1 2 keepfiles |& tee d3samp1-2.log

The output files from this partitioning are *d3samp1.2*, the plot database *parpltA*, and the log file *d3samp1-2.log*. To view the partitioned model, use GRIZ.

griz4s –i parplt	The root name is used for the input to GRIZ.

mv parpltA plt2A	Save the plot file for later viewing. Notice the new plot file name *must* end with *A*.

Step 3. Examine the load balance statistic in the log file

grep "Load balance:" d3samp1-2.log

```
Edge cuts:     360,  Load balance:  1.00000,  Score: 1.18519.
```

Step 4. Repartition the model for four processors.

dynapart d3samp1 4 again keepfiles |& tee d3samp1-4.log
mv parpltA plt4A
grep "Load balance:" d3samp1-4.log

```
Edge cuts:    1080,  Load balance:  1.00000,  Score: 1.55556.
```

Step 5. Repartition the model for eight processors.

dynapart d3samp1 8 again keepfiles |& tee d3samp1-8.log
mv parpltA plt8A
grep "Load balance:" d3samp1-8.log

```
Edge cuts:    2696,  Load balance:  1.02058,  Score: 2.40741.
```

Note that the load balance is very close to 1.000, but the edge cuts increase as the number of processors is increased. Edge cuts are an indication of interprocessor communication. Minimizing interprocessor communication results in more efficient computation.

Example 2. Partitioning a model with contact surfaces

In this example there are ten sliding interface definitions. Six of the interfaces are type 2 sliding interfaces and four are type 3 sliding interfaces. In early versions of ParaDyn, each slideline needed to be fully contained in one processor. In versions 6.1 and later of ParaDyn, large type 2 sliding interfaces can be divided across processors if necessary. Similarly, large type 3 sliding interfaces can be divided across processors if necessary.

We partition the model for 8, 16, 25 and 32 processors.

Table 2 Partitioning and statistics for the df8m3 model

Commands for partitioning the df8m3 model	Comments and statistics
dynapart df8m3 8 keepfiles |& tee df8m3-8.log **grep "Load balance:" df8m3-8.log** **grep "Max # partitions" df8m3-8.log**	Eight processor partition. Load balance: 1.03006. The maximum number of processors is 25.
dynapart df8m3 16 again keepfiles |& tee df8m3-16.log **grep "Load balance:" df8m3-16.log**	Sixteen processor repartition. Load balance: 1.02816.
dynapart df8m3 25 again keepfiles |& tee df8m3-25.log **grep "Load balance:" df8m3-25.log**	Twenty-five processor partition. Load balance: 1.03176.
dynapart df8m3 32 again keepfiles |& tee df8m3-32.log **grep "Load balance:" df8m3-32.log**	Thirty-two processor partition. Load balance: 1.03286.

When partitioning for 8 processors, the software reports that the maximum number of processors for an efficient partition is 25. Since all the large sliding interfaces in this model can be divided across processors, the load balance remains good even for the 32-processor partition. Note that if all sliding interfaces can be divided (as they can in this model), then the estimated maximum number of processors for an efficient partition will increase as the number of processors in the DynaPart command line increases.

In spite of the good load balance statistics for 32 processors, speedup curves for this problem on several platforms show that it should be run on either 8 or 16 processors.

3.4 File Name Sequences

The ParaDyn program (when restarted) and the post-processing software rely on a knowledge of the naming convention for the families of files generated for each processor in a parallel simulation. The default root names of restart files, plot databases, and text files generated in a ParaDyn run include a processor number in the string. Each processor generates a set of files equivalent to a DYNA3D run on a single processor but with a different root name as shown in Table 2. For text files, most of the results are placed in processor 0's file. For a problem running on 1000 or more processors, the length of the string appended to the root name is adjusted to accommodate the

number of digits contained in the number of processors used. For example, for 1024 processors, four digits are added to the root name. Similarly for 10240 processors, five digits are added to the root name.

An example showing the file naming conventions including the Mili state and time history databases is shown in Table 2. These are the default names generated for Mili database files for a problem with less than 1000 processors.

Table 2. Database family names generated by ParaDyn, base name *dynin*

1. State and time history databases: Root names dynin.plt and dynin.th			
P_0	**P_1**	**P_2**	**P_3**
dynin.plt000A	dynin.plt001A	dynin.plt002A	dynin.plt003A
dynin.plt00001	dynin.plt00101	dynin.plt00201	dynin.plt00301
...	...	...	...
dynin.plt00099	dynin.plt00199	dynin.plt00299	dynin.plt00399
P_0	**P_1**	**P_2**	**P_3**
dynin.th000A	dynin.th001A	dynin.th002A	dynin.th003A
dynin.th00001	dynin.th00101	dynin.th00201	dynin.th00301
...	...	...	...
dynin.th00099	dynin.th00199	dynin.th00299	dynin.th00399
2. Restart database names: Root names dynin.dmp1r, dynin.dmp2r, etc.			
P_0	**P_1**	**P_2**	**P_3**
dynin.dmp1r000p	dynin.dmp1r001p	dynin.dmp1r002p	dynin.dmp1r003p
dynin.dmp1r000p01	dynin.dmp1r001p01	dynin.dmp1r002p01	dynin.dmp1r003p01
dynin.dmp1r000p02	dynin.dmp1r001p02	dynin.dmp1r002p02	dynin.dmp1r003p02
3. Text output file names			
P_0	**P_1**	**P_2**	**P_3**
dynin.hsp	dynin.hsp001p	dynin.hsp002p	dynin.hsp003p
dynin.slfor	dynin.slfor001p	dynin.slfor002p	dynin.slfor003p

3.5 ParaDyn Execute Line Options

This section discusses the form of the ParaDyn execute line. On most parallel computers the execute line is preceded by a system utility for copying the executable and execute line over to the processors. Specific examples of this are discussed in the sections on running ParaDyn interactively and under a batch processor.

The ParaDyn execute line options are identical to the DYNA3D execute line. We encourage even experienced users to read Section 3.2 of the DYNA3D manual. There are three options that are important in a parallel simulation: ***q=***, ***l=***, and ***c=***. These options are discussed below and illustrated in the examples. The restarts for ParaDyn likewise use the same execute line as those used in executing a DYNA3D restart.

The execute lines for a ParaDyn initial run and a restart execution are illustrated in the following two examples. The comments relate to the use of the ***q=*** *(or* ***cputime****=),* ***l=***, and ***c=*** *options* which are described below.

Example 1:

paradyn in=*inf* **l=***filelength*	Specify the file length in Mbytes
paradyn in=*resinf* **r=***rtf* **l=***filelength*	Specify the file length on a restart run

Example 2:

paradyn in=*inf* **q=***seconds*	Allow time for file clean up in a batch run
paradyn c=*inf.lastdump* **q=***seconds*	Read restart dump name from file *inf.lastdump*

The first set of execute lines uses the standard input lines for restarts described in the DYNA3D manual. The second set of execute lines illustrates a restart method which starts from the last successfully written restart file.

Notice that it is not necessary to specify an input file for a restart execution, as shown in the second example. If the input file is not specified, all input options but the ***l=*** option will remain the same as those specified in the preceding run. These values from the previous run are stored in the restart dump file and read in during the input phase of the restart run.

The ***l=*** option provides a method for increasing the maximum file length for each member of a database family of plot files. The size specified is in units of Megabytes. The default maximum file length for ParaDyn is 10 Megabytes. *Caution: The file length specified on an initial ParaDyn run must match the file length specified on all subsequent restarts. If a different file length is specified on any restart, ParaDyn will issue an error message and then terminate.*

The ***q=*** *(or* ***cputime=****)* option provides an important capability for terminating a problem run under batch and providing some extra time for file cleanup. The value, in seconds, specifies the number of seconds (by the wall clock) to run the ParaDyn simulation before ParaDyn stops itself with a normal termination. This option is often used to allow the analyst to specify a wall-clock termination time for a batch run that is shorter than the batch time they select. The extra time allows for final file writes and close operations for the database families and other output files.

The ***c=*** option is used to get the name of the last *successfully* written restart file. ParaDyn updates a file named *infile.lastdump* with this restart file name after each write is successfully completed. This restart file is then selected by typing **c=***infile.lastdump*.

The family of output files is named *infile*.hsp, *infile*.hsp001,...., where *infile* is the base name of the input file. Since release 8.1, the program will open any existing *hsp* file and append to it, thus conveniently preserving this information.

Example: Execute lines for ParaDyn

Consider an initial ParaDyn run using the DYNA3D input file, *d3samp1*and a restart run using an input data file, *res*. The initial run and restart run are executed with the following ParaDyn input arguments and files are generated following the naming convention in Table 2.

paradyn in=d3samp1 l=10	File lengths are 10 Mbytes
paradyn in=res r=d3samp1.dmp1r l=10	Restart includes the **l= 10** option.

Notice, the file length is specified and must be equal on both the initial run and on the restart.

This next example illustrates the use of the **q=** *(or* ***cputime*****=***)* termination option.This problem will be submitted to batch to run for one hour (3600 seconds). Ten minutes before the end of the run (after 3000 seconds) ParaDyn will call the termination routine to close files and exit. The maximum length of the files in the plot databases is set to 100 Megabytes. The name of the restart file is the name listed on the first line of the file named *lastdump*.

paradyn in=d3samp1 q=3000 l=100	Allow 10 minutes for file cleanup, 100 Mbyte files, and stop after 3000 seconds
paradyn c=d3samp1.lastdump q=3000 l=100	Restart uses the *lastdump* file, 100 Mbyte files, allows 10 minutes for file cleanup, and stop after 3000 seconds

3.6 Running ParaDyn Interactively

The ParaDyn program is usually run under a system utility, such as **mpirun, prun, poe,** or **srun** on a parallel computer. This system utility copies the ParaDyn executable to the processors on a parallel computer. Generally a small number of processors (2 to 16 processors) are available on a *debug* partition for interactive job processing. This gives code developers and analysts an opportunity to develop and test small prototype models prior to running a large model. It is also useful to run scalability studies on a debug partition if the partition is large enough. Once the model is tested, large ParaDyn simulations are submitted for batch processing. Script files are often made available in public file systems to automate batch runs.

3.6.1 Running ParaDyn Interactively on an IBM AIX system

ParaDyn can be run interactively using the **poe** command on the IBM AIX computers at LLNL. In the examples below, the IBM system has eight processors per node and five hundred or more nodes. The **poe** utility runs ParaDyn with an input specifying the number of nodes and optionally, the total number of processors. The options for specifying the number of nodes and processors are inserted between the ParaDyn command and the execute line options for ParaDyn. If only one processor per node is being used (wasting the rest of the processors on each node), then it is sufficient to simply specify the number of nodes with **-nodes** as follows:

poe paradyn -**nodes** *numnodes paradyn_execute_line*

More than one processor per node can be used by specifying the number of nodes with -**nodes** and the total number of processors with -**procs**.

poe paradyn -**nodes** *numnodes* -**procs** *np paradyn_execute_line*

Example: Interactive ParaDyn runs using poe

The following are command lines for executing ParaDyn with the *d3samp1* input file and using either eight or nine processors. The **-rmpool pdebug** option specifies the interactive pool.

poe paradyn -**nodes 8** -**rmpool pdebug i=d3samp1** Use 1 processor on each of 8 nodes.
poe paradyn -**nodes 1** -**procs 8** -**rmpool pdebug i=d3samp1** Use 8 processors on 1 node.
poe paradyn -**nodes 2** -**procs 9** -**rmpool pdebug i=d3samp1** 9 processors on 2 nodes.

Note that most IBM computers at Livermore Computing (uP and Purple) have eight processors per node.

3.6.2 Running ParaDyn Interactively on a Commodity Linux Cluster

Interactive runs on the Linux clusters at LLNL use the utility **srun** for executing ParaDyn.

srun -**N** *numbernodes* -**n** *np* -**p pdebug paradyn** *paradyn_execute_line*

where *numbernodes* is the number of nodes, and *np* is the number of processors.

Example: Interactive ParaDyn runs using srun

srun -**N 4** -**n 8** -**p pdebug paradyn i=d3samp1** Use 8 processors on 4 nodes (2 procs/node)

The **-p pdebug** option instructs srun to use the pdebug (interactive) pool.

3.6.3 Running ParaDyn Interactively on an SGI Origin

ParaDyn is run interactively using the **mpirun** utility on an SGI Origin computer. The **mpirun** utility runs ParaDyn for *np* processors as follows.

mpirun -**np** *np* **paradyn** *paradyn_execute_line*

where *np* is the number of processors.

Example: Interactive ParaDyn runs using mpirun

mpirun -**np 4 paradyn i=d3samp1** Use 4 processors.

3.7 Running ParaDyn with Batch

ParaDyn production simulations are usually run under the batch system. To set up a problem for batch, a script file is prepared and includes the execute lines for ParaDyn with one difference. The the number of nodes and processors is usually specified on a separate line in the script file rather than included as a keyword argument on the execute line. Thus, the execute line needed in the batch script file includes the utility for setting up the parallel execution followed by the standard paradyn execute line. As an example, the script file to run ParaDyn on an IBM AIX machine includes separate lines for specifying the number of nodes (the -**l nodes=** option in **msub**) and the number of processors per node (the -**l cpn=**option in **msub**). The following are the batch equivalent of the interactive IBM execute lines given in Section 3.6

 # msub -**l nodes=8**
 # psub -**l cpn=1**
poe paradyn i=d3samp1 Use 1 processor on each of 8 nodes.

 # msub -**l nodes=1**
 # msub -**l cpn=8**
poe paradyn i=d3samp1 Use 8 processors on 1 node.

To set up a problem for batch processing at LLNL, prepare a script file and submit it to the batch system with the **msub** utility. Lines included in a typical script file for a ParaDyn simulation on an IBM AIX machine are shown in Figure 8. Lines included in a typical script file for a ParaDyn simulation on a linux cluster are shown in Figure 9. Lines included in a typical script file for a ParaDyn simulation on an IBM BG/P machine (e.g., dawn) are shown in Figure 10.

The script file is submitted to batch for processing using the PSUB utility.

msub *scriptfile*

Use the **mjstat** utility for interrogating the status of runs submitted with **msub**. A status of RUN indicates the job is running on the parallel computer. The online **man** pages can be viewed to study other options provided by the **msub** and **mjstat** utilities. The **mjstat** command displays the status of the job queues and also has **man** pages.

```
# MSUB -j oe # Place both standard output and standard error into one file
# MSUB -l walltime=2:00:00 # Job is to run for a maximum of 2 hours (7200 seconds)
# MSUB -l nodes=4 # 4 nodes
# MSUB -l cpn=8 # 8 processors/node, 32 processors
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
# Allow 10 minutes (600 seconds) for file cleanup
poe paradyn i=dynin q=6600 l=100
echo "ended at"
date
```

Figure 8. A typical batch script file for a ParaDyn simulation on an IBM AIX machine

```
# MSUB -j oe # Place both standard output and standard error into one file
# MSUB -l walltime=2:00:00 # Job is to run for a maximum of 2 hours (7200 seconds)
# MSUB -l nodes=4 # 4 nodes, 32 processors
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
# Allow 10 minutes (600 seconds) for file cleanup
srun -n 32 paradyn i=dynin q=6600 l=100
echo "ended at"
date
```

Figure 9. A typical batch script file for a ParaDyn simulation on a linux cluster.

```
# MSUB -j oe # Place both standard output and standard error into one file
# MSUB -l walltime=2:00:00 # Job is to run for a maximum of 2 hours (7200 seconds)
# MSUB -l nodes=8 # 8 nodes, 32 processors
# MSUB -q pbatch
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
# Allow 10 minutes (600 seconds) for file cleanup
mpirun -np 32 -mode vn -exe /usr/gapps/mdg/sles_10_ppc64/bin/paradyn -cwd ‘pwd‘ -
args ‘i=dynin q=6600 l=100’
echo "ended at"
```

Figure 10. A typical batch script file for a ParaDyn simulation on an IBM BG/P machine. .

3.8 Running Coupled ParaDyn/GEMINI Simulations

ParaDyn has been fully coupled with GEMINI. GEMINI is a parallel Euler solver developed by the Naval Surface Weapons Center, Indian Head Division. It is available at Livermore and other limited sites. GEMINI is an independent executable that runs on a subset of the total set of processors allocated to a job. A coupled ParaDyn/GEMINI run has processors allocated exclusively to ParaDyn, and processors allocated exclusively to GEMINI. ParaDyn processors communicate with GEMINI processors via message passing. The command line for running a ParaDyn/GEMINI coupled simulation is different from a ParaDyn-only simulation. The two programs and their arguments are specified via the command line. On a Linux cluster, a separate file is created that specifies the number of processors and command line for each program. A sample batch file script for running a coupled ParaDyn/GEMINI simulation on a linux cluster, using 64 nodes and 512 processors is shown in Figure 11. The executable line references the file `tasks.480.32`. The contents of that 2-line text file are:

0-479 /usr/apps/mdg/bin/geminimpi
480-511 /usr/apps/mdg/bin/paradyn i=dyn_gem l=100

which contain the execution lines for both ParaDyn and GEMINI. In this example, there are 480 processors allocated to GEMINI (processors 0-479), and 32 processors allocated to ParaDyn (processors 480-511). The equivalent batch file script for running the same coupled ParaDyn/GEMINI simulation on a IBM machine, using 64 nodes and 512 processors is shown in Figure 12.

```
# MSUB -j oe # Place both standard output and standard error into one file
# MSUB -l walltime=2:00:00 # Job is to run for a maximum of 2 hours (7200 seconds)
# MSUB -l nodes=64 # 64 nodes, 512 processors
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
srun -n 512 --multi-prog tasks.480.32
echo "ended at"
date
```

Figure 11. Batch script file for a coupled ParaDyn/GEMINI simulation on a Linux cluster.

To invoke the GEMINI coupling option, add the following to the keyword-based control section of the ParaDyn input file:

gemini_couple 2
couplefile *couple_file_name*

where *couple_file_name* is the name of the file that contains the segments coupled to GEMINI. GEMINI controls ParaDyn's restart file and plot file intervals, termination time and CPU termination time, so it is recommended to disable those quantities on the ParaDyn side.

```
# MSUB -j oe # Place both standard output and standard error into one file
# MSUB -l walltime=2:00:00 # Job is to run for a maximum of 2 hours (7200 seconds)
# MSUB -l nodes=64# 64 nodes
# MSUB -l cpn=8 # 8 processors/node, 512 processors
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
poe -pgmmodel mpmd tasks.480.32
echo "ended at"
date
```

Figure 12. A batch script file for a coupled ParaDyn/GEMINI simulation on an IBM.

3.9 Using Multirun on Dawn

On some parallel computers, the minimum number of processors per job is larger than what is needed. For example, on dawn, an IBM BG/P machine at LLNL, the minimum number of nodes that can be allocated for a batch job is 128. For each node on dawn, 4 processes can be run. This yields a minimum number of processors per job to be 512. If the desired number of processors for a job is much less than 512, one has two choices. 1) Run the small job using the 512 processors, wasting the balance of processors, or 2) use the multirun feature and simultaneously run other jobs on those idle processors.

The multirun feature runs several ParaDyn simulations in one parallel job. The multirun feature is invoked if the file multidyna3d.inp is in the default directory. The format for the text file miltidyna3d.inp is the following:

number of processors for the first job
directory for the first job, relative to the current directory
execute line for the first job
...

number of processors for the last job
directory for the last job, relative to the current directory
execute line for the last job

The total number of processors specified in multidyna3d.inp must br equal to the number of processors in the ParaDyn execute line.

An example batch script to invoke the multirun feature using 512 processors on an IBM BG/P (e.g., dawn) is shown in Figure 13. An example multidyna3d.inp file that invokes 4 128-processor jobs for 512 processors is shown in Figure 14.

A ParaDyn job using the multirun feature will complete after the last subjob is complete. If a subjob crashes, the whole ParaDyn job crashes. The multirun feature should be used only to avoid wasting processors. If each of the subjobs can be run separately without wasting processors, consider submitting separate jobs for each of the subjobs.

```
# MSUB -j oe # Place both standard output and standard error into one file
# MSUB -l walltime=2:00:00 # Job is to run for a maximum of 2 hours (7200 seconds)
# MSUB -l nodes=128 # 128 nodes, 512 processors
# MSUB -q pbatch
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
# Allow 10 minutes (600 seconds) for file cleanup
mpirun -np 512 -mode vn -exe /usr/gapps/mdg/sles_10_ppc64/bin/paradyn -cwd ‘pwd‘
echo "ended at"
date
```

Figure 13. A typical batch script file for a multiriun simulation on an IBM BG/P machine. .

```
128
job1
i=job1 q=6600
128
job2
i=job2 q=6600
128
job3
i=job3 q=6600
128
job4
i=job4 q=6600
```

Figure 14. Contents of multidyna3d.inp for a 4-job multirun job using 512 processors.

3.10 Visualizing ParaDyn Results

Mili binary database format is selected when running ParaDyn simulations. The format for the Mili databases provides a considerable amount of flexibility to the code developers for designing material templates that display results specific to each material type as well as the flexibility for selecting (with input keywords) which state data fields should be included in a database generated by a parallel analysis. In addition, the Mili databases have been extensively tested in large benchmarks and in production runs at both Livermore and Los Alamos.

3.10.1 Combining Parallel Databases and Visualizing Results with GRIZ

Before the binary databases from a ParaDyn simulation can be visualized with GRIZ, the processor families must be combined with the utility Xmilics. Then there are two choices for visualizing the results with GRIZ:

1) The combined databases can be viewed on the parallel computer with GRIZ.
2) The combined database families can be transported (using FTP or SCP) to a workstation and viewed with GRIZ on the workstation.

There are some trade-offs to be made in selecting either of these choices for visualization. For large problems with hundreds of thousands or millions of elements, the rendering and the display of a frame sent over a network to a workstation can be as long as twenty or thirty minutes. In this case better interactivity is possible transporting the combined database over to a local workstation. The additional resources needed for this improved interactivity are: 1) twice the disk space is needed on the parallel computer, 2) a workstation is needed with high-speed graphics capabilities, and with enough disk space to store the full database from a parallel simulation. Even with this additional hardware, the viewing of results is delayed by the time it takes to SCP or FTP the database files to the workstation.

The utility Xmilics combines the Mili state and time history databases from a ParaDyn run. The combined databases can then be viewed with GRIZ.

The execute line for Xmilics for both state and time history databases is:

xmilics -i *infile*

The first argument is the root name for the families of state databases. The output file name is *infile_c*. If one or more files already exist with *infile_c* as their base name, xmilics will query the user if the plot states contained within *infile* should be appended to the existing *infile_c*.

For more information on Xmilics, see APPENDIX 4.

Example: Xmilics execute line

xmilics -i dsamp1.plt	Combine the state database files.
xmilics -i dsamp1.th	Combine the time history database files.

On the workstation or the parallel computer, these databases are viewed with GRIZ. Once GRIZ is started up with a Mili database, the GRIZ *load* command can be used to load another Mili database for viewing.

Example: View a State and Time History Database with GRIZ
View the combined Mili time history and state databases processed with Xmilics in the previous example.

griz4s –i dsamp1.plt_c	Read and view the state plots.
griz4s -i dsamp1.th_c	Read and view time history database

3.10.2 Visualizing Results with VisIt

VisIt is a parallel visualization tool that supports Mili format databases. VisIt can be run on the parallel computer directly or it can be run in a client/server mode using a workstation or PC as the client and the parallel computer as the server. The VisIt Web page at http://www.llnl.gov/visit/about.html includes a *Getting Started* manual as well as a thorough full document. Interactive help information is available with pull down menus in VisIt.

A preprocessing step to VisIt is needed once for each set of Mili databases that will be viewed with VisIt. Use the following VisIt execution to preprocess a Mili database with root name *miliroot:*

visit -makemili *miliroot*	Preprocess Mili database for viewing with VisIt

This will produce a file named *miliroot.mili* which is used by VisIt to open the Mili database family. Once this step is taken, start up VisIt without an argument and open the Mili database by selecting the file *miliroot.mili*.

Example: Visualize the Mili database *d3samp1.plt* **with VisIt.**

visit -makemili d3samp1.plt	Create the *d3samp1.plt.mili* file for VisIt
visit	Start up VisIt for the *d3samp1.plt*Mili database; Open *d3samp1.plt.mili* in the VisIt file panel.

3.10.3 Visualizing Results with EnSight

EnSight is a commercial parallel visualization tool. The installation of EnSight at LANL includes a Mili file reader. The Xmilics combining step is not needed when using EnSight.

3.11 A Summary of Steps for Running ParaDyn Simulations

Set the UNIX **path** variable to the location for the ParaDyn software in the .cshrc file so that it is included each time a new C shell is started.

set path=(/usr/apps/mdg/bin $path)

The documentation is located in the LLNL directory */usr/apps/mdg/doc*.

A summary of steps for executing ParaDyn is shown as a set of steps outlined in tables on the next page.

Steps for Running ParaDyn

Step 1. Partition the mesh and find the optimal number of processors for the run

mkdir partition ; cd partition	Make a directory for partitioning.
ln -s ../dynin .	Link the input file into this directory.
dynapart dynin 2 keepfiles |& tee logfile.2	Run the first partition.
grep "Load Balance:" logfile.2	Check the load balance.
grep "Max # partitions" logfile.2	Check the processor limit.
dynapart dynin4 again keepfiles|& tee logfile.4	Repartition the model.
mv dynin.4 ../ **cd ..**	Move the best partition file to the working directory and change back to the work directory.

Step 2. Execute ParaDyn initial and restart runs

mkdir rundir; cd rundir	Make a directory for the simulation.
ln -s ../dynin . **ln -s ../dynin.4 .**	Link the input and partition files into the directory.
paradyn i=dynin l=100	Initial ParaDyn run with 100 Mbyte file lengths.
paradyn i=res r=dynin.dmp1r l=100	First ParaDyn restart. New input options in file res.
paradyn i=res c=dynin.lastdump l=100	Second restart. Use the restart file name in *lastdump* and input options in file *res.*
paradyn c=dynin.lastdump l=100	Third restart with no resetting of input options.

Step 3. Visualize the results with GRIZ

xmilics -i dynin.plt	Combine the Mili state databases.
xmilics -i dynin.th	Combine the Mili time history databases.
ftp myworkstation	Move the combined databases to a workstation.
griz4s –i dynin.plt_c **griz4s -i dynin.th_ct**	View the results of the state database and the time history database.

This page intentionally blank.

4.0 INPUT FOR PARALLEL SIMULATIONS

This chapter describes data input and special instructions used for parallel simulations only. Instructions for running multiple ParaDyn analyses and special instructions for post-processing results written to text files are described here.

4.1 Tips for Designing Models with Efficient Parallel Contact

Although ParaDyn can be highly efficient on massively parallel computers, performance of a ParaDyn calculation can be severely degraded by the excessive use of certain slidesurface types and other features. This potential degradation can be a result of load imbalance and/or excess interprocessor communication. This can be mitigated by considering the following tips:

Tip 1. Keep the largest slidesurface small enough to fit in one processor, if possible.
This is especially important for slidesurface types 1 and 4. Nodes and elements associated with a slidesurface of these types are assigned to one processor, regardless of the size of the slidesurface. Therefore, the minimum computational time per time step cannot be less than the time required for one processor to process these nodes and elements. This lower bound on the computational time per time step limits the maximum speedup possible, and therefore limits the maximum number of processors that can be efficiently used for the calculation.

Tip 2. Keep slidesurfaces of types 2, 3, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15 and 16 as small as possible if subdivided among processors.
A slidesurface of these types is distributed among as many processors as is necessary during partitioning to generate an efficient partition for the model. Distributing the surfaces causes extra interprocessor communication which lowers performance. Many small slidesurfaces developed during mesh generation can yield better parallel performance than one large slidesurface that is subdivided by partitioning. Furthermore, the small slidesurfaces can be executed on different processors simultaneously which increases the parallelism.

Tip 3. Avoid slidesurface types 1 and 4.
A type 1 slidesurface requires that all associated nodes be contained in one processor. If two slidesurfaces of these types share a node, both slidesurfaces must be computed in the same processor. This can lead to an accumulation in one processor of several type 1 slidesurfaces which have just a few common nodes. This can potentially cause a large load imbalance.

A type 4 slidesurface requires that all associated nodes be contained in one processor. A large type 4 slidesurface could create a load imbalance. Type 12 contact is a good substitute for type 4 contact.

Tip 4. If possible, select user-generated surfaces for automatic contact.
By default ParaDyn generates the segments (surface patches) that are used for automatic contact types 12 and 14. However, in some cases the analyst may have defined and optimized the contact surface segments for the model. In this case surface segments can be specified manually for the automatic contact interfaces by using the keyword *segments*. This keyword is added in the keyword input section of the automatic contact interface definition.

segments 1
User-defined slave and master segments are specified. Segments associated with shell elements are treated as shell elements, with thickness, by the automatic contact algorithm.

segments 2 and **segments 3**
User-defined slave and master segments specified. Segments associated with shell elements are treated as brick elements, without thickness, by the automatic contact algorithm, and have an effective depth = (*segments)* x **(***thickness)*. This option generates behavior similar to type 3 contact, and can be more robust than the **segments 1** option.

Tip 5. If using automatic segment generation in automatic contact, use domain limiting keywords to reduce the size of the generated contact surface.
Performance may improve if the size of a slidesurface is reduced with domain-limiting keywords. These keywords are added in the keyword input section of the automatic contact interface definition. This list of keywords and values is documented in the DYNA3D User's Manual[1].

xmin, xmax, ymin, ymax, zmin, zmax keywords
Specifiy domain limits for the automatic-contact search.

mat_in mat#1 mat#2 mat#3 … mat#n keyword input
Specify material numbers to include in automatic-contact segment generation.

mat_ex mat#1 mat#2 mat#3 … mat#n keyword input
Specify material numbers to exclude in automatic-contact segment generation.

normal_include mat# px py pz _min _max keyword input

Include only those segments whose angle between the vector from the center of segment to the point (px, py, pz) and the segment normal are between *_min* and *_max* degrees.

4.2 Multiple Versions of Running Restart Files

The following describes a capability for generating multiple versions of a running restart dump file. The option is selected by specifying in the keyword-based control section of the input file:

numrrf *nvers*

where *nvers* specifies the number of different versions of the running restart file to save on disk. The default value of *nvers* is 2. The running restart family names are incremented through a set of family members until nvers of them have been written. Once this limit is reached the next running restart over writes the first running restart file in the set. The examples below illustrate the cycling through of the names of the versions of the running restart files.

The number of cycles between the running restart dump files is selected as usual in the fifth data field, columns 36-40, of DYNA3D control card 6. It can also be set by specifying in the keyword-based control section of the input file:

ncbrrf *n*

where *n* specifies the number of cycles between running restart files.

The names for the running restart file for ParaDyn are *infile*.**rdmpr***nnn***p***mmm*, where *infile* is the input file base name, *i* is the restart number, *nnn* is the three-digit processor number and *mmm* is the family-member number.

Example: Specify multiple versions of the running restart file

Set up the input file to save three different versions of the running restart file, writing a restart file every 30 cycles

numrrf 3 ncbrrf 30 endfree

Suppose this problem is run for 100 cycles, remembering that the number of cycles between running restart dumps is 30. The files generated for a 4-processor run will be:

in.rdmp1r000p in.rdmp1r001p in.rdmp1r002p in.rdmp1r003p (cycle 30)
in.rdmp2r000p in.rdmp2r001p in.rdmp2r002p in.rdmp2r003p (cycle 60)
in.rdmp3r000p in.rdmp3r001p in.rdmp3r002p in.rdmp3r003p (cycle 90)

Suppose this problem is run for another 150 cycles, then the running restart files will be

in.rdmp1r000p in.rdmp1r001p in.rdmp1r002p in.rdmp1r003p (cycle 120)
in.rdmp2r000p in.rdmp2r001p in.rdmp2r002p in.rdmp2r003p (cycle 150)
in.rdmp3r000p in.rdmp3r001p in.rdmp3r002p in.rdmp3r003p (cycle 180)
in.rdmp1r000p in.rdmp1r001p in.rdmp1r002p in.rdmp1r003p (cycle 210)
in.rdmp2r000p in.rdmp2r001p in.rdmp2r002p in.rdmp2r003p (cycle 240)

In the above, the cycle (30, 60, 90) files were overwritten by the cycle (120, 150, 180) files, respectively. And finally, the cycle (210, 240) files overwrite the cycle (120, 150) files, respectively. After cycle 240, the files will be

in.rdmp1r000p in.rdmp1r001p in.rdmp1r002p in.rdmp1r003p (cycle 210)
in.rdmp2r000p in.rdmp2r001p in.rdmp2r002p in.rdmp2r003p (cycle 240)
in.rdmp3r000p in.rdmp3r001p in.rdmp3r002p in.rdmp3r003p (cycle 180)

5.0 RELEASE NOTES

Significant Revisions from Version 10.1

Mechanics

- Select hypo-elastic material models can now be run with hourglass controls types 8 and 10 for improved stability.
- The reference temperature for material model 4 is now prescribed with a keyword.

Contact

- In automatic contact, SAND active boxes now permanently ignore nodes and segments whenever they do not reside in any active boxes.

Input/Output

- Arbitrary node and element numbers can be used to specify nodes and elements, respectively. There is no limitation to compact, 1-to-*n* orderings.

Architectural revisions

- The pressure-only SPH feature was incorporated into the main code architecture. Input information is now specified with keywords, and SPH output is incorporated into the regular plot files.

New or Enhanced Mechanics Features

Boundary Conditions

- None.

Contact

- Added the ability to specify friction coefficients by material pairs for automatic contact.
- Each slide surface side now tests for and removes duplicate contact segments.
- Frictional forces can now be capped at a user prescribed value.
- The `consider_free` automatic contact feature can be turned off at restart time.
- A mortar-based, kinematic enforced, tied contact type, 17, has been added.
- A self-generation feature to create interior slave nodes has been added, e.g., for type 5

- Only one `.slfor` file is written when printed slide surface output is requested.
- An option to better deal with sharp edges has been added to type 16 mortar contact. The new feature allows multiple slave node pressures at a single node.

Material Models

- A hyper-elastic elasto-plastic material model for solid elements has been added - #73.
- The shell constitutive routine for material model 12 was changed to use an implicit plane-stress projection algorithm.
- Material models 33 and 34 were revamped and now function as described in the DYNA3D manual.
- The initialization of material models that employ a separate material coordinate system were unified and a new element-specific method to specify the local axes was added.

Mechanics

- A new 3-D discrete element formulation was added with broader features.
- The communication routines used for SPH were altered to reduce computational cost and improve resolution.
- Meshless particles have been improved to better handle spalled material points when using material model 10.

Input/Output Options

- Node and element print block input options have been expanded. The new options also allow users to better control what data they want included in time history files.
- ParaDyn now checks for non-consecutive node and element numbers, and permits them only when the appropriate version of DynaPart has been used to generate the partition file.
- The control card keyword `restart_plotfiles` restarts the Mili time history and plot databases at state 0 to allow restarts runs to be launched with only the Mili “A” files.
- A new element plot variable named “cause” has been added to indicate the reason why an element is deleted during a calculation.
- Material point strains can optionally be included in the plot database for hex, meshless, and nodal tet elements.

New or Enhanced Parallel Features

ParaDyn Upgrades

- All new or enhanced mechanics features are available in parallel.
- Writing to the HSP file has been consolidated, and for a normal run only the main file is created.
- Only one `.slfor` file is written when printed slide surface output is requested.

DynaPart Upgrades

- Added support to optionally use parallel PARSKMETIS to do graph partitioning when running DynaPart in a parallel job. Optional command line keywords *parskmetis*, *partcommcost, partminloadbaltarget, partmaxloadbaltarget, partmaxrefines, parttimes, partinitseed, partsummaryverb, partloopsverb, partprinttiming, nopartprinttiming* and *parallelruncommand* now available.
- Checks to see if DynaPart was run using an extension to the command name. If so, append that same extension to DYNA3D to read the ASCII input file and write the binary input file. For example, if the command name was *dynapart.alpha*, use *dyna3d.alpha* to parse the input file.
- Introduce command line keyword *parserpath*. If *parserpath* is supplied, the supplied pathname specifies which version of DYNA3D to use to parse the input file, rather than using the algorithm above.

DynaPart Architectural Revisions

- Append information from the labels file generated by the DYNA3D input parser to the partition file.

Resolved Defects

Mechanics

- Fixed a problem that caused the wrong material mass to be written for parts made from nodal tets.
- Reworked the momentum deposition algorithm so it works correctly in parallel.
- Resolved a formulation issue with how material model 63 treated thermal strains.
- Fixed an incorrect sign in one of the higher order terms in material model 27.

Contact

- Removed a numerical noise issue in type 16 mortar contact that caused master segments that are perpendicular to a slave segment to be incorrectly handled.
- Fixed the calculation of slide-surface tractions included in the regular plot files. The algorithm contained errors and was not parallelized.
- When running with a fixed time-step size, penalty contact can no longer modify the time-step size. If contact thinks a smaller time-step size should be used, a warning message is issued.
- The internal reordering of incorrectly prescribed triangular contact segments sometimes caused memory problems. This problem has been resolved.
- Type 9 contact was incorrectly reporting nodal relocation when relocation was turned off due to an incorrect format statement.

Input/Output

- Time history plot files for the transient phase were being initialized incorrectly when time history files were requested for the dynamic relaxation phase.
- Keyword support was added for the nodal coordinate format variable.
- The SPH birth certificate file is now being opened correctly at restart time.
- The time interval between writing of print data (and time history files) is no longer set to a big number whenever it is larger than the termination time.
- Time history files written during dynamic relaxation did not contain a state with the initial configuration.

DynaPart

- On IBM BG/P systems, DynaPart can be run as a serial job.
- Upgrade the SKMETIS graph partitioning software to version sk-09. This version computes the partition score based on the number of vertices rather than on the total graph weight. It also applies some bug fixes.

Known Restrictions and Unresolved Defects

Element Features

- None.

Contact

- Type 1 sliding only contact influences the master side beyond that directly below the slave side.

Output Options

- None.

DynaPart

- PFGEN always reports in the log file that the number of materials is zero.

ParaDyn

- None.

This page intentionally blank

REFERENCES

1. Lin, Jerry I. and Zywicz, E., "DYNA3D: A Nonlinear, Explicit, Dimensional Finite Element Code for Solid and Structural Mechanics—User Manual," Lawrence Livermore National Laboratory, Livermore, California, UCRL-MA-107254, **2010**.
2. Speck, D. E., Dovey, D. J., "GRIZ: Finite Element Analysis Results Visualization for Unstructured Grids—User Manual," Lawrence Livermore National Laboratory, Livermore, California, UCRL-MA-115696, **2000**.
3. Corey, I.R., Pierce, E. and Speck, D. E., "MILI 1.0: A Mesh I/O Library—Programmer's Reference Manual", Lawrence Livermore National Laboratory, Livermore, California, UCRL-SM-236144, **2007**.
4. Sherwood, Robert J., "DynaPart Auxiliary Documentation Files", Lawrence Livermore National Laboratory, Livermore, California, UCRL-ID-137888, **2000**.
5. Schauer, D. A., Hoover, C.G., Kay, G. J., Lee, A.S., and De Groot, A.J., "Crashworthiness Simulations with DYNA3D", Paper No. 961249, Transportation Research Board, **1996**.
6. Karypis, G. and Kumar, V., "METIS 3.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System," University of Minnesota, Department of Computer Science, **1997**. See also http://www-users.cs.umn.edu/~karypis/metis/main.shtml.
7. Karypis, G. and Kumar, V., "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359--392, **1999**.
8. Karypis, G. and Kumar, V., "Multilevel k-way Partitioning Scheme for Irregular Graphs," Journal of Parallel and Distributed Computing, **1997**.
9. Hendrickson, B. and Leland, R., "An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations," Sandia National Laboratory Report Number SAND92-1460, **1992**.
10. Hoover, C.G., Badders, D. C., De Groot, A.J., and Sherwood, R. J., "Parallel Algorithm Research for Solid Mechanics Applications Using Finite Element Analysis," Lawrence Livermore National Laboratory, Livermore, California, UCRL-ID-129202, **1997**.
11. A.J. DeGroot, R.M. Ferencz, M.A. Havstad, N. Hodge, J. Lin, I.D. Parsons, M.A. Puso, R. Sherwood, J.M. Solberg and E. Zywciz: "Accomplishments and Challenges in Code Development for Parallel and Multimechanics Simulation," pp. 216–227 in High-Performance Computing for Computational Science VECPAR 2008 (eds. J. Palma, P. Amestoy, M. Dayde, M. Mattoso and J. Lopes), Springer-Verlag, **2008**.

This page intentionally blank.

APPENDIX 1. DynaPart Command Line and Keywords

As discussed in Section 3, a DYNA3D input model must be partitioned before using it as input to ParaDyn. The partition file generated by DynaPart includes the lists of nodes and elements assigned to each processor. This file is read by the ParaDyn program and is the map used to distribute the model across the processors on a parallel computer.

1.1 Running the DynaPart Software

To access the MDG software, the standard path name must be in the Unix PATH variable. C shell users working on the Livermore Computing systems may add this line to their .cshrc file to set the PATH variable when a new C shell is started.

set path = (/usr/apps/mdg/bin $path)

Online help information that includes the format of the DynaPart command line and descriptions of all of the keywords is available by typing:

dynapart -help

The DynaPart script requires two or more arguments. The first two arguments are mandatory, and the remainder are optional keyword arguments. The first argument is the name of the DYNA3D input file and second argument is the number of processors over which you wish to distribute this job. The syntax of the dynapart command is:

dynapart <*dyna-input-file*> <*number-processors*> *[keyword...]*

Items in <> are required. Items in [] are optional.

For example, if the name of the DYNA3D input file is *bigmodel* and you wish to run on 32 processors, cd to the directory containing file bigmodel and issue the following command:

dynapart bigmodel 32

The above command will generate a partition file named bigmodel.32, and will also generate diagnostic output to the screen. It is very useful to save this diagnostic output for future reference. If you wish to save the diagnostic output to file bigmodel32.log, then replace the command above with:

dynapart bigmodel 32 |& tee bigmodel32.log

Carefully look through the screen output (or the file bigmodel32.log) to make sure that each phase of the partitioning was successful. If error messages were issued, correct the DYNA3D input file and run DynaPart again.

When the partitioning runs to a successful completion, the partition file will be in the current directory, and will have a name of the form <*job-name*>.<*number-of-processors*>. In the example above, the partition file generated is bigmodel.32.

Intermediate files are generated during the partitioning process. If you wish to retain those files for later diagnostic purposes or for a subsequent faster repartitioning of the model, use the *keepfiles* command line keyword:

dynapart bigmodel 32 keepfiles |& tee bigmodel32.log

For large models that have long DynaPart run times, it is often possible to repartition the model for a different number of processors using the intermediate files generated in a prior run of DynaPart. This can be done if (1) the DYNA3D input file has not changed since the previous partition, (2) the intermediate files from the previous partition were saved in the current working directory by using command line keyword *keepfiles*, (3) those intermediate files have not been modified, and (4) you have not changed the state of command line keywords *badelemabort, nobadelemabort, smallgraph, nosmallgraph, segbased, nosegbased, optslide, nooptslide, optslideradius, opslideweight, optslidebuckets* or *parserpath*. For a definition of these keywords, see Table 3. The shorter DynaPart run is selected using the keyword *again* on the DynaPart command line. Using this keyword saves time because it eliminates redundant executions of several DynaPart programs:

dynapart bigmodel 64 keepfiles again |& tee bigmodel64.log

When using the *again* keyword, you assume all responsibility to see that the DYNA3D input file has not changed, the above keyword values have not changed, and all of the DynaPart intermediate files from the previous partition of this same job remain unmodified since the earlier run. If there is any doubt, do not use the *again* keyword.

The Mili plot file parpltA is overwritten when DynaPart is run again in the same directory. The plot file can be saved by renaming it before partitioning the model another time. The last character in the new name must continue to be an A.

1.2 Optional Keyword Arguments to DynaPart

There are a number of keyword options that affect the partitioning of an input file. Some keywords modify numerical parameters that are used during partitioning, and other keywords modify the partitioning algorithms in other ways. Recall that the syntax for the dynapart command is:

dynapart <*dyna-input-file*> <*number-processors*> *[keyword...]*

Items in <> are required. Items in [] are optional.

There can be any number of keywords, including zero. If no keywords are specified, all keywords assume their default values. For most models, using the dynapart command without any keywords will result in a high quality partition. We recommend that you initially partition without specifying keywords, then later use keywords only if a low quality partition results when running dynapart without keywords. Until you need to use DynaPart keywords, you can skip the remainder of Appendix 1. Keywords can be specified in any order, except that keywords that specify a value must be followed immediately by that value.

The list of optional keywords in DynaPart is shown in Table 3.

Table 3. DynaPart Keywords

Keyword	Function
again	Rerun DynaPart using intermediate files from the previous run.
badelemabort	Print warning and abort if a bad element is read. Default.

Table 3. DynaPart Keywords

Keyword	Function
nobadelemabort	Print warning but do *not* abort if a bad element is read.
smallgraph	Generate a small graph without edge weights.
nosmallgraph	Generate a standard graph with edge weights. Default.
segbased	Use segments where appropriate to calculate Special Element sets. Default.
nosegbased	Use nodes rather than segments to calculate Special Element sets.
maxsettoconsider <m.n.>	Maximum ratio of the cost of a separable set to the cost of the entire graph. Larger separable sets will *not* receive special handling. Range is 0.1 to 1.0, default is 0.8.
optslide	Optimize sliding interfaces by adding edges to the graph. Default.
nooptslide	Do *not* optimize sliding interfaces by adding edges to the graph.
optslideradius <n>	Do *not* add a graph edge between segments of a sliding interface if they are connected within this distance in segments. Default is 3.
optslideweight <n>	Use edge weight of *n* when adding edges between segments on a sliding interface. Default is 5.
optslidebuckets <n>	Divide the search space for each sliding interface into *n* buckets in each direction to optimize the search for nearest neighbors in the contact search algorithm. Default is 20.
setcostmult <m.n>	Cost multiplier for metavertices. Default is 1.5.
maxsetfillfactor <m.n>	Max size for separable-set segment in units of processor capacity. Range is 0.1 to 1.0, default is 0.9.
dice	Subdivide separable sets too large for one processor. Default.
nodice	Do *not* subdivide large separable sets.
metis306	Use Metis V3.0.6 rather than the default Metis.

Table 3. DynaPart Keywords

Keyword	Function
kmetis	Run kmetis instead of default skmetis.
pmetis	Run pmetis instead of default skmetis.
parskmetis	Run parskmetis instead of default skmetis.
skmetis	Run skmetis. Default.
multiconstraint	Run multiconstraint Metis instead of default single constraint.
partitioncommcost <m.n>	Communication cost for edge cut in skmetis. Default is 0.5.
partitiontimes <n>	Number of times to partition for each set of skmetis parameters. Default is 4.
partitionverbosity <n>	Amount of status information from skmetis: 1 = minimum 2 = each parameter set 3 = each iteration. Default is 1.
partcommcost <m.n>	Communication cost factor for parskmetis edge cut. Default is 0.2.
partminloadbaltarget <m.n>	Minimum load balance target for parskmetis. Default is 1.01.
partmaxloadbaltarget <m.n>	Maximum load balance target for parskmetis. Default is 1.06.
partmaxrefines <n>	Maximum number of refinements for parskmetis. Default is 50.
parttimes <n>	Number of times to partition for each parskmetis parameter. Default is 4.
partinitseed <n>	Initial random number seed for parskmetis. Default is 0.
partsummaryverb <n>	Verbosity for printing parskmetis summary statistics: 0 = do not print summary statistics for partitions 1 = print statistics for partition with best score 2 = also print some statistics for best and worst partitions 3 = also print all statistics for best and worst partitions. Default is 2.

Table 3. DynaPart Keywords

Keyword	Function
partloopsverb <n>	Verbosity for printing parskmetis loops statistics: 0 = do not print statistics for each partition 1 = print some statistics for each partition 2 = also print some statistics for each refinement 3 = also print all statistics for each partition. Default is 0.
partprinttiming	Have parskmetis print its timing information.
nopartprinttiming	Do not have parskmetis print its timing information. Default.
parallelruncommand <string>	String for the parallel run command, including the number of processors, enclosed in quotes (for example 'srun -n32'). This must be specified if parskmetis is to run in parallel mode. Default is blank (run as a serial job).
plot	Generate a Mili plot file of the partition. Default.
noplot	Do *not* generate a plot file of the partition.
keepfiles	Keep the intermediate files for use later with keyword *again*, or for use in diagnosing partitioning problems.
nokeepfiles	Do *not* keep the intermediate files. Default.
parserpath <path>	Use the executable at <path> to parse the ASCII input file and write the binary input file. Default is dyna3d.<extension_used_to_run_dynapart>.

The following is a discussion of each of the keywords.

again

This keyword eliminates redundant initial steps when repartitioning a data set that has been partitioned earlier. It may *not* be used if (1) the DYNA3D input file has been changed, (2) the intermediate files from the previous partition were *not* saved in the current working directory by using command line keyword *keepfiles*, (3) those intermediate files have been modified, or (4) the status of keywords *badelemabort, nobadelemabort*, *smallgraph, nosmallgraph, segbased*, *nosegbased*, *optslide, nooptslide, optslideradius, optslideweight, optslidebuckets* or *parserpath*

have been changed. It may be used if changing the number of partitions or the values of other keywords. If there is any doubt, do *not* use the *again* keyword. If *again* is not specified, the default is to repeat all steps of the partitioning.

badelemabort
Cause SNPGEN to issue a warning message then abort after reading the input file if a bad element was found. A bad element for this purpose is defined to be an element with an invalid node number or a degenerate element. A degenerate element is an element with an insufficient number of unique nodes. A hexagonal element is degenerate if it contains less than four unique nodes or if it contains a segment with less than three unique nodes. A shell or thick shell element is degenerate if it contains less than three unique nodes. A beam (or truss) element is degenerate if it contains less than two unique nodes (excluding the reference node). This is the default.

nobadelemabort
Cause SNPGEN to issue a warning message but *not* abort after reading the input file if a bad element was found.

smallgraph
Generate a small graph. This option reduces the size of the graph by using a more restrictive definition for touching elements and by not generating edge weights. Using this option discards some useful information and can result in a less efficient partition. This option should only be used if a very large model fails to partition due to insufficient memory for DUALGEN or SKMETIS.

nosmallgraph
Generate a standard graph. This option uses a more inclusive definition for touching elements, and generates edge weights. This is the default.

segbased
Use segments (also known as facets) to generate sets of elements that must be handled in a single processor. Elements are included only if they contain all four nodes of a segment in a special DYNA3D object such as a sliding interface. This is the default.

nosegbased
Use nodes to generate sets of elements that must be handled in a single processor. Elements are included if they contain any node in a special DYNA3D object such as a sliding interface.

maxsettocconsider <m.n>

Maximum ratio of the cost of a separable set to the cost of the entire graph. Larger separable sets will not receive special handling. Range 0.1 to 1.0, default 0.8.

optslide

Program LINGRF will optimize sliding interfaces by adding edges to the graph. These edges connect nearest neighbor segments in the sliding interface which are assumed to be on opposite sides of the sliding interface. This is the default.

nooptslide

Do *not* optimize sliding interfaces by adding edges to the graph.

optslideradius <n>

When adding edges between segments on a sliding interface, do *not* add an edge if the proposed segment is connected to this segment within a distance of *n* segments. Default is 3.

optslideweight <n>

When adding edges between segments on a sliding interface, set the edge weight to *n*. Default is 5.

optslidebuckets <n>

To optimize the search for nearest neighbors in sliding interfaces, subdivide the search space into *n* buckets in each direction. Default is 20.

setcostmult <m.n>

Specify a floating point cost multiplier for special sets. This multiplier is a floating point number of value greater than 1.0 to represent the extra computational burden imposed by DYNA3D objects generating special sets (such as sliding interfaces). Default is 1.5.

maxsetfillfactor <m.n>

Maximum size for a segment of a separable set, in units of processor capacity. If *dice* is specified, program CUTSEP will cut large separable sets into segments that are no larger than this. The value for *maxsetfillfactor* should be between 0.1 and 1.0, and defaults to 0.9. This is ignored if keyword *nodice* is used.

dice

Program CUTSEP will subdivide (dice) separable sets if they are too large for one processor. If a separable set is too large to fit in one processor, DynaPart will determine how many segments it needs to be cut into, and will cut it optimally. This is the default.

nodice

Do *not* subdivide large separable sets, even if they are too large for one processor. If such large separable sets are present, the resulting partition will have a suboptimal load balance.

metis306

At the heart of the partitioning software is a graph partitioning program called Metis, from the University of Minnesota. The latest release of Metis is Version 4 or higher, but very few users may prefer partitions that are performed by Metis Version 3.0.6. Specifying keyword *metis306* will cause DynaPart to run Version 3.0.6 of pmetis or kmetis. If you do *not* specify *metis306*, DynaPart will default to using the most recent version of Metis on the system.

The four metis keywords specifying a Metis program (*kmetis*, *pmetis*, *skmetis* or *parskmetis*) are mutually exclusive. Only one may be specified. If none are specified, *skmetis* is used.

kmetis

Use kmetis rather than skmetis, parskmetis or pmetis. Program kmetis uses the k-way partitioning algorithm.

pmetis

Use pmetis rather than skmetis, parskmetis or kmetis. Program pmetis uses the recursive partitioning algorithm. This is the default if metis306 is specified.

skmetis

Use skmetis rather than parskmetis, kmetis or pmetis. Program skmetis runs several variations of the kmetis routine, then selects the result having the best score based on both load balance and edge cuts. This option takes more time and memory to partition than *pmetis* or *kmetis* do, but may result in a better partition. This is only available in Metis Version 4 and higher. This is the default unless *metis306* is specified.

parskmetis:

Use parskmetis rather than skmetis, kmetis or pmetis. Program parskmetis is a parallel routine that runs several variations of the ParMetis K-way partitioning algorithm followed by K-way refinements, then selects the result having the best score based on both load balance and edge cuts. When running inside a parallel job, this routine can use multiple processors to make more efficient use of the resources than skmetis, kmetis or pmetis can. It can also partition larger graphs than the others can. To run on more than one processor, keyword *parallelruncommand* must be specified. See below.

multiconstraint
Run multi-constraint Metis. Use at your own risk. This will result in more time spent running Metis, and is only available if *metis306* is *not* specified. Multi-constraint Metis may produce a better partition. The *multiconstraint* option may not be used with parskmetis. The default is to run single-constraint Metis.

partitioncommcost <m.n>
Program skmetis computes the quality of a partition based on both load balance and the number of graph edge cuts (which correspond to interprocessor communications). Keyword *partitioncommcost* is used to specify a real number that is the multiplier of the number of edge cuts when computing the partition score. It is normally between 0.0 and 1.0, but may be larger. Value 0.0 means edge cuts are not considered in the score. Default is 0.5.

partitiontimes <n>
Program skmetis varies several parameters and computes partition(s) for each set of parameter values. For a given set of parameters, partitions are computed *partitiontimes* times, with a different state of the random number generator for each calculation. After all partitions with all values of parameters have been computed, the partition with the best score is returned. Default is 4.

partitionverbosity <n>
The amount of reporting from skmetis can have one of three values:

partitionverbosity 1	1 = Minimum reporting. Report the statistics and the control parameters that generated the best partition.
partitionverbosity 2	2 = Medium reporting. In addition to (1), for each value of control parameters, report summary statistics (ranges of load balances and edge cuts) for the partitions that were computed.
partitionverbosity 3	3 = Full reporting. In addition to (2), report the statistics (load

balance and edge cuts) for *each* partition that was calculated. Default is 1.

partcommcost <m.n>
Program parskmetis computes the quality of a partition based on both load balance and the number of graph edge cuts (which correspond to interprocessor communications). Keyword *partcommcost* is used to specify a real number that is a multiplier of the number of edge cuts when computing the partition score. It is normally between 0.0 and 1.0, but may be larger. Value 0.0 means edge cuts are not considered in the score. Default is 0.2.

partminloadbaltarget <m.n>
Specify the minimum load balance target for parskmetis. Program parskmetis varies the target load balance from the minimum value specified to the maximum value specified, in steps of 0.01. In general, partitions generated with a low target load balance will have a lower (better) load balance but will have a higher (worse) number of edge cuts. Default is 1.01.

partmaxloadbaltarget <m.n>
Specify the maximum load balance target for parskmetis. Default is 1.06.

partmaxrefines <n>
Specify the maximum number of refinements for each partition computed by parskmetis. Multiple K-way refinements are done on each partition until either (1) the number of edge cuts is no longer reduced, or (2) the maximum number of refinements is reached. Default is 50.

parttimes <n>
Program parskmetis varies the target load balance and computes partitions for each target load balance. For each target load balance, partitions are computed *parttimes* times, with a different seed for the random number generator for each calculation. After all partitions have been computed, the partition with the best score is returned. Default is 4.

partinitseed <n>
Specify the initial random number generator seed to be used by parskmetis. Default is 0.

partsummaryverb <n>
Specify the verbosity for the summary statistics printed by parskmetis.

0 = do not print summary statistics for partitions

1 = print statistics for the partition with the best score
2 = also print some statistics for the best and worst partitions
3 = also print all statistics for the best and worst partitions.
Default is 2.

partloopsverb <n>
Specify the verbosity for intermediate partition statistics printed by parskmetis.
0 = do not print statistics for each partition
1 = print some statistics for each partition
2 = also print some statistics for each refinement
3 = also print all statistics for each partition.
Default is 0.

partprinttiming
Have parskmetis print the amount of time that it used in various phases of partitioning.

nopartprinttiming
Do not have parskmetis print the amount of time that it used in various phases of partitioning. This is the default.

parallelruncommand <quoted string>
Specify the string that is used to run a parallel job on your system, including the number of processors/cores enclosed in quotes (for example 'srun -n32'). This is used to run parskmetis if you used keyword *parskmetis*. To run using 32 processors/cores on a CHAOS system, the string 'srun -n32' is a possibility. To run using all the processors/cores assigned to this parallel job on a CHAOS system, the string 'srun' is a possibility. To run using all the processors/cores assigned to this parallel job on a SuSE SLES 10 system, 'mpirun -mode VN' is a possibility. Modify as required for your system. Default is blank (run as a serial job).

plot
Generate a Mili format plot file of the partition. This is the default.

noplot
Do *not* generate a plot file of the partition.

keepfiles

Keep the intermediate files written by DynaPart for later use. Use this keyword if you intend to repartition later using keyword *again*. Also use this keyword if you want to diagnose a partitioning problem.

nokeepfiles:
Do not keep the intermediate files written by DynaPart. This is the default.

parserpath <path>:
Use the executable at <*path*> to read the ASCII input file and write the binary input file. Path <*path*> must be an absolute pathname (must contain the full directory path and the executable name (for example /usr/apps/mdg/bin/dyna3d.alpha)). If the *parserpath* keyword is not specified, the default parser is dyna3d.<extension_used_to_run_dynapart>. For example, dynapart.beta will default to running dyna3d.beta to do the parsing.

Table 4. DynaPart Keyword Defaults and Overrides

Parameter	Default	Override with keyword
Faster repartition?	Rerun all	*again*
Abort if bad element is seen?	Yes	*nobadelemabort*
Use small graph to reduce memory?	No	*smallgraph*
Segment-based contact?	Yes	*nosegbased*
Maximum size for separable set	0.8	*maxsettoconsider <m.n>*
Add graph edges across sliding interfaces?	Yes	*nooptslide*
Minimum segment distance for adding a graph edge.	3	*optslideradius <n>*
Weight of each added graph edge.	5	*optslideweight <n>*
Number of search buckets in each direction for finding graph edges to add.	20	*optslidebuckets <n>*
Set-cost multiplier for meta-vertices.	1.5	*setcostmult <m.n>*
Maximum separable-set chunk	0.9	*maxsetfillfactor <m.n>*
Subdivide large separable sets?	Yes	*nodice*

Table 4. DynaPart Keyword Defaults and Overrides

Parameter	Default	Override with keyword
Metis version	Latest	*metis306*
Metis program	skmetis	*parskmetis, kmetis, pmetis*
Multiconstraint Metis?	Single	*multiconstraint*
skmetis communication cost	0.5	*partitioncommcost <m.n>*
skmetis # times each parameter set	4	*partitiontimes <n>*
skmetis verbosity	1 (low)	*partitionverbosity <n>* 2 (medium), 3(full)
parskmetis communication cost	0.2	*partcommcost <m.n>*
parskmetis min load balance target	1.01	*partminloadbaltarget <m.n>*
parskmetis max load balance target	1.06	*partmaxloadbaltarget <m.n>*
parskmetis max # refinements	50	*partmaxrefines <n>*
parskmetis # times each target	4	*parttimes <n>*
parskmetis initial random seed	0	*partinitseed <n>*
parskmetis print summary verbosity	2	partsummaryverb <n> 0 = none 1 = best 2 = best and worst short 3 = best and worst long
parskmetis print loops verbosity	0	partloopsverb <n>: 0 = none 1 = some each 2 = add refinements 3 = add all statistics for each
parskmetis print timing info?	No	*partprinttiming*
parskmetis parallel run command	(none)	*parallelruncommand <quoted string>*
Plot file?	Yes	*noplot*
Keep intermediate files?	No	*keepfiles*
Use default input parser?	Yes	*parserpath <path>*

1.3 Using the *dice/nodice* and *optslide/nooptslide* Keywords

The following are guidelines to help the analyst decide whether to use the *dice* or *nodice* keywords, and the *optslide* and *nooptslide* keywords.

- If not specified, the software takes the default of *dice* and *optslide*.
- The *optslide* keyword causes program LINGRF to be run. LINGRF enhances the graph representing the model by adding graph edges across sliding interfaces to help generate a partition with lower communication (a faster partition).
- "Separable sets" are generated by sliding interfaces of type 2, 3, 5, 6, 7, 8, 9, 10, 12, 13 and 14.
- If there are no large separable sets present, simply default (do not specify) the *dice* keyword. There is no harm in using *dice* (the default). The software run by the *dice* option will have no effect in this situation.
- If there are one or more large separable sets present that correspond to automatic contact sliding interfaces, then:
  a). *dice* should be used to subdivide the large separable sets, (however *dice* is the default, so does not need to be specified).
  b). *optslide* should be used to optimize sliding interfaces, (however, *optslide* is the default, so does not need to be specified).

1.4 Example DynaPart command lines

1. Partition input file *bigmodel* for 32 processors, using all the default parameters:

dynapart bigmodel 32 |& tee bigmodel32.log

2. Partition input file bigmodel for 32 processors, keeping the intermediate files with the intention of later repartitioning for a different number of processors:

dynapart bigmodel 32 keepfiles |& tee bigmodel32.log

3. Repartition *bigmodel* for 64 processors after partitioning it previously and without changing the input file, intermediate files, or other keywords:

dynapart bigmodel 64 again |& tee bigmodel64.log

4. We attempted to partition a very large model in a batch job on the largest computer. The model was so large that DUALGEN failed due to insufficient memory. Try generating a smaller graph, even though the partition may be slightly less optimal:

dynapart bigmodel 32 smallgraph |& tee bigmodel32.log

5. Partition *bigmodel* for 32 processors, without using segments:

dynapart bigmodel 32 nosegbased |& tee bigmodel32.log

6. If a sliding interface contains more than 80% of the model (in terms of computational cost), do not give it special consideration during partitioning:

dynapart bigmodel 32 maxsettoconsider 0.8 |& tee bigmodel32.log

7. Partition *bigmodel* for 32 processors, but don't optimize sliding interfaces by adding edges to the graph:

dynapart bigmodel 32 nooptslide |& tee bigmodel32.log

8. Partition *bigmodel* for 32 processors, but multiply the estimated compute load to compute sliding interfaces and other special element sets by a factor of 1.83:

dynapart bigmodel 32 setcostmult 1.83 keepfiles |& tee bigmodel32.log

9. We discovered from an earlier DynaPart run that even though the largest special element set was from a separable set, the load balance still was higher than desired. Now we want to repartition, making the largest such set only 0.8 of a processor's worth:

dynapart bigmodel 32 dice maxsetfillfactor 0.8 again |& tee bigmodel32.log

10. Partition *bigmodel* for 32 processors. If separable sets are too large for one processor, subdivide them as necessary:

dynapart bigmodel 32 dice |& tee bigmodel32.log
or
dynapart bigmodel 32 |& tee bigmodel32.log

11. Partition *bigmodel* for 32 processors, but do not subdivide large separable sets (such as automatic contact sliding interfaces), even if they are too large for one processor:

dynapart bigmodel 32 nodice |& tee bigmodel32.log

12. Partition *bigmodel* for 32 processors, using kmetis Version 3.0.6:

dynapart bigmodel 32 metis306 kmetis |& tee bigmodel32.log

13. Partition *bigmodel* for 32 processors, using pmetis:

dynapart bigmodel 32 pmetis |& tee bigmodel32.log

14. Partition *bigmodel* for 32 processors, using multiconstraint skmetis:

dynapart bigmodel 32 multiconstraint |& tee bigmodel32.log

15. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis run on 32 processors/cores on a CHAOS system (modify as required for your system):

**dynapart bigmodel 32 parskmetis parallelruncommand 'srun -n32' **
|& tee bigmodel32.log

16. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis run on all processors/cores assigned to this job on a CHAOS system (modify as required for your system):

**dynapart bigmodel 32 parskmetis parallelruncommand 'srun' **
|& tee bigmodel32.log

17. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis run on all processors/cores assigned to this job on a SuSE SLES 10 system (modify as required for your system):

**dynapart bigmodel 32 parskmetis parallelruncommand 'mpirun -mode VN' **
|& tee bigmodel32.log

18. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis run on one processor/core per node assigned to this job on a SuSE SLES 10 system. This is not recommended unless parskmetis is running short of memory (modify as required for your system):

**dynapart bigmodel 32 parskmetis parallelruncommand 'mpirun' **
|& tee bigmodel32.log

19. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis with a partition communication cost factor of 0.1, run on all processors/cores assigned to this job on a CHAOS system:

d**ynapart bigmodel 32 parskmetis partcommcost 0.1 **
parallelruncommand 'srun' |& tee bigmodel32.log

20. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis with a minimum load balance target of 1.03, run on all processors/cores assigned to this job on a CHAOS system:

**dynapart bigmodel 32 parskmetis partminloadbaltarget 1.03 **
parallelruncommand 'srun' |& tee bigmodel32.log

21. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis with a maximum number of refinements per partition of 10, run on all processors/cores assigned to this job on a CHAOS system:

**dynapart bigmodel 32 parskmetis partmaxrefines 10 **
parallelruncommand 'srun' |& tee bigmodel32.log

22. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis computing 8 partitions for each target load balance, run on all processors/cores assigned to this job on a CHAOS system:

**dynapart bigmodel 32 parskmetis parttimes 8 **
parallelruncommand 'srun' |& tee bigmodel32.log

23. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis with an initial random number seed of 50, run on all processors/cores assigned to this job on a CHAOS system:

**dynapart bigmodel 32 parskmetis partinitseed 50 **
parallelruncommand 'srun' |& tee bigmodel32.log

24. Partition input file bigmodel for 32 processors in a parallel job, using parskmetis and printing some statistics for each partition, run on all processors/cores assigned to this job on a CHAOS system:

**dynapart bigmodel 32 parskmetis partloopsverb 1 **
parallelruncommand 'srun' |& tee bigmodel32.log

25. Partition *bigmodel* for 32 processors, and generate a Mili format plot file:

dynapart bigmodel 32 plot |& tee bigmodel32.log

26. Partition *bigmodel* for 32 processors, but do not generate a plot file:

dynapart bigmodel 32 noplot |& tee bigmodel32.log

27. Run dynapart.beta, but use dyna3d.alpha to do input parsing:

**dynapart.beta bigmodel 32 parserpath /usr/apps/mdg/bin/dyna3d.alpha **
|& tee bigmodel32.log

APPENDIX 2. DynaPart Log File

This appendix provides more detailed description of the background processing in DynaPart. It is intended to expand on the examples provided in Section 3.

Up-to-date information about the latest changes to the production version of DynaPart is available online in the standard shared documentation directory. The files related to DynaPart in this directory are:

- DYNAPART-HOW-TO-PARTITION: This provides instructions for running DynaPart, including a complete list of keywords, their meanings, default values, and examples.
- DYNAPART-FOR-GOOD-PARTITIONS: This lists the DYNA3D objects which constrain partitioning, and how to interpret DynaPart log files.
- DYNAPART-PARTITION-FILE-FORMAT: This documents changes to the partition file format.
- DYNAPART-RELEASE-NOTES: These release notes describe differences from previous versions of the software (text format).
- DYNAPART-RELEASE-NOTES.pdf: These release notes describe differences from previous versions of the software (PDF format).

Up-to-date information about later versions of DynaPart (for example the alpha or beta versions) is available in the dedicated documentation directory associated with that version. The file names in that directory do not have the leading "DYNAPART-" prefix. For example, the PDF version of the release notes is RELEASE-NOTES.pdf.

2.1 DynaPart Log File

A complete log file from a DynaPart execution is shown below. The two statistics of interest to the analysts running the problem are highlighted with underlined, bold text. These statistics are (1) the maximum number of processors (partitions) for good computational load balance for contact and other special options printed by program REDUCEGRF and (2) the computational load balance

printed by program SKMETIS or PARSKMETIS. The programs run by the DynaPart script depend on the mesh and boundary conditions for the specific problem being partitioned, as explained in APPENDIX 3. The programs executed in DynaPart for this particular problem are highlighted with double-underlined text.

The first part of the log file includes the following items:

- The DynaPart options selected either by default or from keywords specified on the execution line. See Appendix 1 for details about keywords.
- A brief description of each of the programs run by the DynaPart script.
- The file extensions for files generated during partitioning. The files generated by DynaPart use the input file name as a root name and use an extension for each type of file generated during the partitioning process.
- Statistics and other output generated by the programs run in the DynaPart script.

The following is an example of the output from each of the programs executed by DynaPart Version 11.1 when it was run as a parallel job on 32 processors and the *parskmetis* option was taken. DynaPart was run with the following execution line, and generated a log file df8m3-32.log:

**dynapart.beta df8m3 32 parskmetis parallelruncommand 'srun' keepfiles **
|& tee df8m3-32.log

DynaPart Log File for problem *df8m3*

```
This program is export controlled.

DynaPart version 11.1, dp110530, $Revision: 1.49 $,
  produced 30-May-2011 at 18:02 PDT, for ParaDyn version 11.1,
  installed in /usr/gapps/mdg/chaos_4_x86_64_ib/archive/dynapartdir/dp110530/
bin,
  and running on host zeus17.

Partitioning df8m3 for 32 processors.

DynaPart keyword arguments: parskmetis parallelruncommand srun keepfiles

DynaPart options:
  Read ASCII input file and write binary input file.
  Use /usr/apps/mdg/bin/dyna3d.beta to write the binary input file.
  Abort if a bad element is read.
  Use a standard graph with edge weights.
  Sliding interfaces of Type 3, 5, 8, 9, 10 will be subdivided if necessary.
```

```
  Sliding interfaces of Type 2, 6 or 7 will be subdivided if necessary.
  Use segments where appropriate to generate Special Element file.
  Optimize sliding interfaces by adding graph edges across them.
    Radius of protected region is 3 segments.
    Weight of edges to add across sliding interfaces is 5.
    Maximum number of buckets in each dimension is 20.
  Subdivide large separable sets (e.g. autocontact sliding interfaces).
    Make each piece no larger than 0.9 of a processor's worth.
  Do not consider separable sets if their cost is more than 0.8 of the
    cost of the entire graph.
  Use current default version of parskmetis.
  Use single-constraint parskmetis.
    Cost of an edge cut by parskmetis is 0.2.
    Vary target load balance from 1.01 to 1.06 in steps of 0.01.
    Refine each partition no more than 50 times.
    Compute 4 partitions for each target load balance.
    Seed the random number generator with 0 for the first partition.
    Use 'srun' to make the parallel run of parskmetis.
  Multiply cost of sets (e.g. sliding interfaces) by 1.5.
  Keep DynaPart intermediate files in the working directory.
  Write a Mili plot file.

This script executes the software in the following order:
  DYNA3D - Reads the DYNA3D input file, extracts the data
           needed by DynaPart, and writes that data to a
           binary file for use by subsequent DynaPart routines.
  SNPGEN - Generates sets of special nodal points, segments,
           SAND special elements, set types, free format keyword
           sections, nodal coordinates, and selected item sizes.
  SELSETS - Selects the ganged sets from the SNPGEN output.
  DUALGEN - Generates a dual graph file, special elements (SE) file,
            enhanced segment file, and vertex weight file.
  LINGRF - Adds edges to the graph across most sliding interfaces,
           but not across ganged (type 1 or 4) sliding interfaces.
  MERGESETS - Merges a SAND elements file into a SE file.
  SETCOST - Computes cost of each SE set.
  DRNBIGSETS - Drains separable sets that contain nearly the entire model.
  CUTSEP - Cuts separable sets that are too large for one processor.
  EXTGRF - Extracts subgraphs from a graph.
  CTS - Converts a color file and its new-to-old map to a SE file.
  TRANSCLOSE - Transitively closes the ganged special elements.
  AETS - Assigns special elements to disjoint sets.
  REDUCEGRF - Reduces the graph by reducing SE sets to a meta-element.
  PARSKMETIS - Partitions the reduced graph, produces a color file.
  EXPANDCOL - Expands the color file to include all original elements.
  SETCOL - Specifies color for all special element sets.
  SELSETS - Selects the ganged and unganged sets from the SNPGEN output.
  SELCOLS - Selects colors for ganged and unganged sets from SNPGEN.
  PFGEN - Generates a partition file.

Internal and output file extensions:
  .dp_bin          Extracted data from DYNA3D input file (from dyna3d)
  .n_labels        Extracted labels from DYNA3D input file (from dyna3d)
  .ssff1           Special Sets Free-Format prelim file (from extinput)
```

```
  .snp              Special Nodal Points (SNP) file (from snpgen)
  .seg              Segment file (from snpgen)
  .sst              Special Sets Type file (from snpgen)
  .ssff             Special Sets Free-Format file (from snpgen)
  .sso              Special Sets Object file (from snpgen)
  .ndb              Node database file (from snpgen)
  .sesand           Special Elements for SAND file (from snpgen)
  .size             Number of SNP sets and elements (from snpgen)
  .gsnp             Ganged Special Nodal Points file (from selsets)
  .grforig          Dual graph file (from dualgen)
  .se               Special Element (SE) file (from dualgen)
  .eseg             Enhanced segment file (from dualgen)
  .vwt              Vertex weight file (from dualgen)
  .grfee            Graph file with additional edges (from lingrf)
  .seorsand         Special element or SAND element file (from mergesets)
  .seorsandcost     Cost of sets in .seorsand file (from setcost)
  .seorsmallsep     Special element or small separable file (from drnbigsets)
  .seorsmallsepcost Cost of sets in .seorsmallsep file (from setcost)
  .bigsep           Oversized separable SE sets (from cutsep)
  .grfsep<M>        Subgraph of vertices from big SE set <M> (from extgrf)
  .nosep<M>         New-to-old vertex map from big SE set <M> (from extgrf)
  .grfsep<M>.part.<N> Local color file (from metis) for big SE set (metis)
  .secut<M>         SE file of big SE set <M> after cutting (from cts)
  .sesep            SE file after big separables cut (from cutsep)
  .sstsep           Special Set Types file after cutting (from cutsep)
  .nosesep          New-Old Special Element set mapping (from cutsep)
  .setcg            Special Elements Transitively Closed Gang file
                      (from transclose)
  .ontcgs           Old-New TransClosed Gang Set mapping file
                      (from transclose)
  .dse              Disjoint Special Elements file (from aets)
  .onses            Old-New Special Element Set mapping file (from aets)
  .grfred           Reduced Graph file (from reducegrf)
  .one              Old-New Element mapping file (from reducegrf)
  .grforig.part.<N> Local color file (from metis)
  .grfred.part.<N> Local color file (from metis)
  .part.<N>         Global color file (from expandcol)
  .snscol           Special Nodal Set Color file (from setcol)
  .uggsnp           SNP file from unganged and ganged only (from selsets)
  .uggsnscol        Color file for SNP sets from unganged/ganged only
                      (from selcols)
  .<N>              Partition file (from pfgen)
 parpltA            Mili-format partition plot file (from pfgen)

Thursday 14-Jul-2011 at 17:01:24 PDT

Running DYNA3D.beta (/usr/apps/mdg/bin/dyna3d.beta)


 ******************************************************************

     *******    **    **  *      **    *****    *********  *******
     ********   **    **  **     **   *******   ********   ********
      **   ***  **    **  ***    **  ***   ***       **     **   ***
```

```
      **     **    **  **    ****    **  **      **       ***     **     **
      **     **     ****     ** **   **  **      **      *****    **     **
      **     **      **      **  ** **  *********         ***    **     **
      **     **      **      **   ****  *********           **    **     **
      **   ***       **      **     ***  **      **   ***   ***    **   ***
     ********        **      **      **  **      **    *******   ********
     *******         **      **       *  **      **     *****    *******

                  #####   #####      #        #
                 #     # #     #    #       ##
                       # #     #   # #     # #
                  #####  #     #    #       #
                 #       #     #    #       #
                 #       #     #    #       #
                 #######  #####   #####   #####


          dyna3d (version 11.1.2   from 06/03/2011 14:09) opt=  6

                       $Revision: 1.1903.2.2 $

 ************************************************************************


 ************************************************************************
 *                                                                      *
 *  ******  *      * *      *     *     ******      *     ******  *******   *
 *  *      * *      * **     *    * *    *      *    * *    *      *     *   *
 *  *      *  *    *  * *    *   *   *   *      *   *   *   *      *     *   *
 *  *      *    ***    *   *  * *      * ******  *      * ******      *   *
 *  *      *     *     *    * * ******* *        ******* *     *      *   *
 *  *      *     *     *     ** *      * *        *      * *      *     *   *
 *  ******      *     *      * *      * *        *      * *      *     *   *
 *                                                                      *
 *                                                                      *
 *               ******  *      *      *     *****  *******             *
 *               *      * *      *     * *    *        *                *
 *               *      * *      *    *   *   *  *     *                *
 *               ******  *******  *      *  *****  *****                 *
 *               *       *      *  *******        * *                   *
 *               *       *      *  *      *       * *                   *
 *               *       *      *  *      *  *****  *******             *
 *                                                                      *
 ************************************************************************


 ************************************************************************
 *                                                                      *
 *                    ******  ******* *******     *                    *
 *                    *     * *           *      * *                   *
 *                    *     * *           *     *   *                  *
 *                    ******  *****       *     *     *                 *
 *                    *     * *           *     *******                 *
```

```
 *                  *     * *          *    *     *                  *
 *                  ******  *******    *    *     *                  *
 *                                                                    *
 ********************************************************************


 n o r m a l    t e r m i n a t i o n
0
1.354u 0.374s 0:02.15 80.0%0+0k 0+0io 72pf+0w

Running SNPGEN

        ****  SNPGEN - Generate a Special Nodal Points file  ****
                        Version of 13-May-2010

 <<  Starting to read the Dyna3d binary input file.

  DIPOLE FACADE 8 (ppp wes-ssa)

           Number of nodal points:                       157331
           Number of 8-node hexagonal elements:          142369
           Number of 2-node beam elements:                 9646
           Number of 4-node shell elements:                   0
           Number of 8-node thick-shell elements:             0

 >>  Finished reading the Dyna3d binary input file.

 --  Starting to sort the special nodal points lists.

 ++  Finished sorting the special nodal points lists.

 --  Starting to write the special nodal points output file.

 ++  Finished writing the special nodal points output file.

 --  Starting to write the segment output file.

 ++  Finished writing the segment output file.

 --  Starting to write the special element SAND output file.

 --  Starting to sort the special nodal points lists.

 ++  Finished sorting the special nodal points lists.

 ++  Finished writing the special element SAND output file.

 --  Starting to write the set type output file.

 ++  Finished writing the set type output file.

  There were:
        10 Special Nodal Point sets.  The longest set contained
      2208 nodal points, and all sets combined contained a total of
```

```
    12690 nodal points.
  There were:
       10 segment sets.  The longest set contained
     2046 segments, and all sets combined contained a total of
    13938 segments.
  There were:
       10 Special Element SAND sets.  The longest set contained
        0 elements, and all sets combined contained a total of
        0 elements.
 Normal Termination
0.009u 0.016s 0:00.08 12.5%0+0k 0+0io 8pf+0w
```

Running SELSETS

```
     ****  SELSETS - Select sets of the specified type(s)  ****
                    Version of  1-Jul-2002

 Selecting sets of type 2.

 << Starting to read the sets input file

 >> Finished reading the sets input file

 << Starting to read the set types input file

 >> Finished reading the set types input file

 -- Starting to write the selected sets output file

 ++ Finished writing the selected sets output file

 There were:
        0 selected sets.  The longest set contained
        0 elements, and all sets combined contained a total of
        0 elements.
0.001u 0.002s 0:00.00 0.0%0+0k 0+0io 1pf+0w
```

Running DUALGEN

```
          ****  DUALGEN - Generate a dual-mesh graph file  ****
                         Version of 13-May-2010

 <<  Starting to read the Dyna3D binary input file.

  DIPOLE FACADE 8 (ppp wes-ssa)

          Number of materials:                              7
          Number of nodal points:                      157331
          Number of 8-node hexagonal elements:         142369
          Number of 2-node beam elements:                9646
          Number of 4-node shell elements:                  0
          Number of 8-node thick-shell elements:            0

 >>  Finished reading the Dyna3D binary input file.
```

```
 <<  Starting to read the set type input file.

 >>  Finished reading the set type input file.

 --  Starting to generate the node-to-element connectivity array.

 ++  Finished generating the node-to-element connectivity array.

 --  Starting to generate Special Element and enhanced segment files.

     These Special Element sets DO share some elements.
     You should run TRANSCLOSE to combine SE sets having shared elements.

 ++  Finished generating Special Element and enhanced segment files.

 --  Starting to generate the element-to-element connectivity array.

 ++  Finished generating the element-to-element connectivity array.

 --  Starting to generate the dual-mesh graph file.

     Maximum vertex weight will be set to    1000.
     Min elem cost at elem    142370 orig cost    1.22 scaled cost     612.
     Max elem cost at elem    120626 orig cost    2.00 scaled cost    1000.

     Total cost of the unscaled graph was           252989.
     Total cost of the scaled graph is           126559852.

 ++  Finished generating the dual-mesh graph file.

 --  Starting to generate the vertex weight file.

 ++  Finished generating the vertex weight file.

     Maximum element connectivity of   56 seen at element  131795.

 Normal Termination
2.362u 0.283s 0:02.95 89.4%0+0k 0+0io 14pf+0w

Running LINGRF

        ****  LINGRF - Link to neighbors in the graph file  ****
                         Version of 10-Jun-2010

    Bucketing being used.  Maximum number of buckets in X, Y, Z: 20 20 20.

 << Starting to read the graph input file

 >> Finished reading the graph input file

 << Starting to read the extended segment input file

 >> Finished reading the extended segment input file
```

```
<< Starting to read the nodal database input file

>> Finished reading the nodal database input file

<< Starting to read the special set type input file

>> Finished reading the special set type input file

-- Starting to compute the segment pairs
   WARNING: In set 6 there were a total of 264 segments that were repeats of
            earlier segments, and 264 unique earlier segments to which they
            were identical.  The elements listed with the later instances
            of the segments will be ignored.
   WARNING: In set 7 there were a total of 648 segments that were repeats of
            earlier segments, and 648 unique earlier segments to which they
            were identical.  The elements listed with the later instances
            of the segments will be ignored.
   WARNING: In set 8 there were a total of 648 segments that were repeats of
            earlier segments, and 648 unique earlier segments to which they
            were identical.  The elements listed with the later instances
            of the segments will be ignored.
   WARNING: In set 9 there were a total of 648 segments that were repeats of
            earlier segments, and 648 unique earlier segments to which they
            were identical.  The elements listed with the later instances
            of the segments will be ignored.

++ Finished computing the segment pairs

-- Starting to compute the extra graph edges

   There were:
       5656 pairs of segments to be linked with one element each,
       2362 pairs of segments with more than one element per segment, and
       4570 total additional elements.
     Only the first element for each segment was considered for an
     extra graph edge.

++ Finished computing the extra graph edges

-- Starting to compute the output graph

++ Finished computing the output graph

   Radius of segment dilation was 3.
   There were originally 1892550 edges in the input graph file.
   There are now 1898148 edges in the output graph file.
   LINGRF added 5598 edges (0.30% additional edges).
   Each added edge had a weight of 5.

-- Starting to write the graph output file

   Output graph statistics:
    Sum of vertex weights:                126559852
```

```
     Largest vertex weight:                      1000
       at vertex:                              120626

 ++ Finished writing the graph output file

2.923u 0.036s 0:03.24 91.0%0+0k 0+0io 3pf+0w

Running MERGESETS

                    ****  MERGESETS - Merge sets  ****
                          Version of  8-Jun-2010

 <<  Starting to read the primary set input file

 >>  Finished reading the primary set input file

 <<  Starting to read the update set input file

 >>  Finished reading the update set input file

 --  Starting to compute the merged sets

 ++  Finished computing the merged sets

 --  Starting to write the merged sets output file

 ++  Finished writing the merged sets output file

                                    Primary Sets   Update Sets   Result Sets
                                    ------------   -----------   -----------
     Number of sets:                          10            10            10
     Largest element in any set:          142369             0        142369
     Number elements in largest set:        3816             0          3816
     Number elements in all sets:          15846             0         15846

0.008u 0.000s 0:00.01 0.0%0+0k 0+0io 1pf+0w

Running SETCOST

               ****  SETCOST - Compute costs of sets  ****
                          Version of 20-Feb-2002

 << Starting to read the special element set input file

 >> Finished reading the special element set input file

 << Starting to read the element weight input file

 >> Finished reading the element weight input file

 -- Starting to write the set costs output file

 ++ Finished writing the set cost output file
```

```
 Set cost statistics:
   Number of sets:                                    10
   Cost of most costly set:                      3129120
   Total cost of all sets:                      14066880
   Total cost of all vertices in graph:        126559852
0.057u 0.000s 0:00.06 83.3%0+0k 0+0io 1pf+0w

Running DRNBIGSETS

                 ****  DRNBIGSETS - Drain big sets  ****
                         Version of  1-Mar-2006

 << Starting to read the set input file

 >> Finished reading the set input file

 << Starting to read the set cost input file

 >> Finished reading the set cost input file

 << Starting to read the set type input file

 >> Finished reading the set type input file

 -- Starting to compute the drained sets

 ++ Finished computing the drained sets

 -- Starting to write the drained sets output file

 ++ Finished writing the drained sets output file

0.003u 0.000s 0:00.01 0.0%0+0k 0+0io 1pf+0w

Running SETCOST

              ****  SETCOST - Compute costs of sets  ****
                          Version of 20-Feb-2002

 << Starting to read the special element set input file

 >> Finished reading the special element set input file

 << Starting to read the element weight input file

 >> Finished reading the element weight input file

 -- Starting to write the set costs output file

 ++ Finished writing the set cost output file

 Set cost statistics:
   Number of sets:                                    10
   Cost of most costly set:                      3129120
```

```
   Total cost of all sets:                          14066880
   Total cost of all vertices in graph:            126559852
0.055u 0.001s 0:00.05 100.0%0+0k 0+0io 0pf+0w
```

Running CUTSEP

```
    CUTSEP will subdivide large separable sets by an additional factor of 1.

        ****  EXTGRF - Extract a subgraph from a larger graph  ****
                          Version of 29-Apr-2010

 << Starting to read the vertex list input file

 >> Finished reading the vertex list input file

 << Starting to read the graph input file

 >> Finished reading the graph input file

 -- Starting to write graph output file df8m3.grfsep

    Output graph statistics:
      Sum of vertex weights:                   3129120
      Largest vertex weight:                       820
      Vertex with largest weight:                    1

 ++ Finished writing graph output file df8m3.grfsep

 -- Starting to write new-to-old vertex map output file df8m3.nosep

 ++ Finished writing new-to-old vertex map output file df8m3.nosep

**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Graph Information ---------------------------------------------------
  Name: df8m3.grfsep, #Vertices: 3816, #Edges: 31376, #Parts: 2

Recursive Partitioning... --------------------------------------------
  2-way Edge-Cut:     876, Balance:  1.00

Timing Information --------------------------------------------------
  I/O:             0.000
  Partitioning:    0.000   (PMETIS time)
  Total:           0.000
**********************************************************************

    ****  CTS - Compute a set file from map and color files  ****
                        Version of 30-Apr-2000

 << Starting to read the new-to-old element mapping file

 >> Finished reading the new-to-old element mapping file
```

```
 << Starting to read the color file

 >> Finished reading the color file

 -- Starting to write the sets output file

 ++ Finished writing the sets output file


    ****  CUTSEP - Find and subdivide the oversize separable sets  ****
                           Version of 17-May-2010

 << Starting to read the special element set input file

 >> Finished reading the special element set input file

 << Starting to read the sets cost input file

 >> Finished reading the sets cost input file

 << Starting to read the sets type input file

 >> Finished reading the sets type input file

    Cost of entire graph is 126559852.
    Number of special elements before being made disjoint is 15846.
    Estimated (probably high) cost of reduced graph is 126567775.
    Number of processors is 32.
    Set cost multiplier is  1.50.
    Maximum set fill factor is  0.90.
    Therefore, cost of largest permissible separable set is 2373145.
    Separable set 10 has cost 3129120.  It will be cut into 2 sets.

 -- Starting to write the new-to-old set mapping output file

 -- Starting to write the big separable special element sets output file

 ++ Finished writing the big separable special element sets output file

 -- Starting to run EXTGRF
 ++ Finished running EXTGRF

 -- Starting to run PMETIS for df8m3.grfsep

 ++ Finished running PMETIS

 -- Starting to run CTS to generate df8m3.secut
 ++ Finished running CTS

 -- Starting to write the special element sets output file

 ++ Finished writing the special element sets output file

 -- Starting to write the special sets type output file
```

```
 ++ Finished writing the special sets type output file

 ++ Finished writing the new-to-old set mapping output file

1.830u 0.030s 0:01.90 97.8%0+0k 0+0io 6pf+0w

Running TRANSCLOSE

      ****  TRANSCLOSE- Perform transitive closure on ganged sets  ****
                          Version of 29-Apr-2010

 <<  Starting to read the sets input file

 >>  Finished reading the sets input file

     There were a total of   15846 values read, in      11 sets.

 <<  Starting to read the set type input file

 >>  Finished reading the set type input file

 --  Starting to calculate transitively closed ganged sets

 ++  Finished calculating transitively closed ganged sets

 --  Starting to write the set trans-closed gang output file

 ++  Finished writing the set trans-closed gang output file

 --  Starting to write the old-to-new set mapping output file

 ++  Finished writing the old-to-new set mapping output file

 There were:
        11 output sets.  The longest set contained
      2046 elements, and all sets combined contained a total of
     15846 elements.

 Normal Termination
0.003u 0.007s 0:00.02 0.0%0+0k 0+0io 9pf+0w

Running AETS

                  ****  AETS - Assign elements to sets  ****
                            Version of 12-Feb-2002

 << Starting to read the transitively closed gang set input file

 >> Finished reading the transitively closed gang set input file

 << Starting to read the sets type input file

 >> Finished reading the sets type input file
```

```
 << Starting to read the old-to-new set mapping input file

 >> Finished reading the old-to-new set mapping input file

 -- Starting to write the disjoint sets output file

 ++ Finished writing the disjoint sets output file

 -- Starting to write the old-to-new set mapping output file

 ++ Finished writing the old-to-new set mapping output file

 There were:
       11 disjoint sets.  The longest set contained
     2024 elements, and all sets combined contained a total of
    15066 elements.
0.004u 0.002s 0:00.21 0.0%0+0k 0+0io 1pf+0w

Running REDUCEGRF

                ****  REDUCEGRF - Reduce a graph file  ****
                          Version of  8-Jun-2010

 << Starting to read the special element input file

 >> Finished reading the special element input file

 << Starting to read the graph input file

 >> Finished reading the graph input file

    There were 7512 connections between meta-vertices.

 -- Starting to sort each row of the output graph

 ++ Finished sorting each row of the output graph

 -- Starting to write the graph output file

    Output graph statistics:
     Sum of vertex weights:                 133240912
     Largest vertex weight:                   2940960
       at vertex:                              136954
       which represents special element set:        5
     Max # partitions for good load balance:       45

 ++ Finished writing the graph output file

2.652u 0.053s 0:02.96 91.2%0+0k 0+0io 1pf+0w

Running PARSKMETIS
******************************************************************************
  PARSKMETIS 3.1.1.sk-01 Copyright Lawrence Livermore National Security, LLC.
```

```
  Based on ParMetis 3.1.1 Copyright Regents of the University of Minnesota.
*****************************************************************************

Partitioning graph file df8m3.grfred using 32 processors/cores.

Started reading input file df8m3.grfred
Completed reading input file df8m3.grfred

Graph Information ---------------------------------------------------------
  Name: df8m3.grfred, #Vertices: 136960, #Edges: 1652741, #Parts: 32.
Scoring and Calculation Information ---------------------------------------
  Min Load Bal Target: 1.010,  Max Load Bal Target: 1.060.
  Edge cut factor: 0.200,  Max number of refinements: 50.
  Number of iterations per parameter set: 4,  Initial seed: 0.
Parallel K-way Partitioning -----------------------------------------------
  A total of 24 partitions were calculated, including 139 refinements.

Partition with best score:
  Load balance:          1.010997      Edge cuts:              427586
  Score:                 1.121376      Edge cut score factor:    0.2
  Number of refinements:        4
  Using load balance target: 1.01      Using random seed:          3

Overview of best and worst partitions based on three metrics
        In general these are six different partitions

             ------------ Best -----------    ----------- Worst -----------
  Metric     Value  #Refines  Target  Seed    Value  #Refines  Target  Seed
  ------     -----  --------  ------  ----    -----  --------  ------  ----
Load bal   1.010237     6      1.01     2   1.061586     7      1.06     2
Edge cuts    400851     7      1.06     2     450902     7      1.01     0
Score      1.121376     4      1.01     3   1.173439     4      1.06     0

0.006u 0.017s 0:17.89 0.0%0+0k 0+0io 9pf+0w

Running EXPANDCOL

                ****  EXPANDCOL - Expand a color file  ****
                          Version of 15-Jan-1999

 << Starting to read the global-local elements input file

 >> Finished reading the global-local elements input file

 << Starting to read the reduced color input file

 >> Finished reading the reduced color input file

 -- Starting to write the color output file

 ++ Finished writing the color output file

0.099u 0.002s 0:00.11 81.8%0+0k 0+0io 1pf+0w
```

```
Running SETCOL

    ****  SETCOL - Generate a color file for the special sets  ****
                         Version of  1-Jul-2002

 << Starting to read the old-to-new set mapping input file

 >> Finished reading the old-to-new set mapping input file

 << Starting to read the reduced graph color input file

 >> Finished reading the reduced graph color input file

 -- Starting to write the set color output file

 ++ Finished writing the set color output file

0.014u 0.000s 0:00.01 100.0%0+0k 0+0io 1pf+0w

Running SELCOLS

 ****  SELCOLS - Select colors for sets of the specified type(s)  ****
                       Version of  7-Apr-2004

 Selecting sets of type 1 or 2.

 << Starting to read the set types input file

 >> Finished reading the set types input file

 << Starting to read the set colors input file

 >> Finished reading the set colors input file

 -- Starting to write the set colors output file

 ++ Finished writing the set colors output file

 There were 0 selected colors.
0.000u 0.000s 0:00.00 0.0%0+0k 0+0io 1pf+0w

Running SELSETS

      ****  SELSETS - Select sets of the specified type(s)  ****
                       Version of  1-Jul-2002

 Selecting sets of type 1 or 2.

 << Starting to read the sets input file

 >> Finished reading the sets input file

 << Starting to read the set types input file
```

```
 >> Finished reading the set types input file

 -- Starting to write the selected sets output file

 ++ Finished writing the selected sets output file

 There were:
         0 selected sets.  The longest set contained
         0 elements, and all sets combined contained a total of
         0 elements.
0.002u 0.001s 0:00.00 0.0%0+0k 0+0io 0pf+0w

Running PFGEN

          ****  PFGEN - Generate a partition-assignment file  ****
                        Version of 26-May-2011

 <<  Starting to read the Dyna3D binary input file.

  DIPOLE FACADE 8 (ppp wes-ssa)

          Number of materials:                              0
          Number of nodal points:                      157331
          Number of 8-node hexagonal elements:         142369
          Number of 2-node beam elements:                9646
          Number of 4-node shell elements:                  0
          Number of 8-node thick-shell elements:            0

 >>  Finished reading the Dyna3D binary input file.

 <<  Starting to read the partition-color file.

          Number of processors in the color file:          32

 >>  Finished reading the partition-color file.

 <<  Starting to read the gang node set file.

 >>  Finished reading the gang node set file.

 <<  Starting to read the pre-assigned node set file.

 >>  Finished reading the pre-assigned node set file.

 --  Starting to generate the partition-assignment file.

     There are no unassigned nodes to assign.

                                 Number of      Number of
                                   Node         Adjacent
                                Communications  Processors

        Average per processor:      2069.9          8.2
         Max on any processor:      3740           19
```

```
      Total on all processors:     66238           264

     Copied 309346 records from the label file.

 ++  Finished generating the partition-assignment file.

 --  Starting to generate the Mili plot file.

 ++  Finished generating the Mili plot file.

 Normal Termination
0.533u 0.254s 0:00.97 80.4%0+0k 0+0io 17pf+0w

Thursday 14-Jul-2011 at 17:01:57 PDT

Partition file name is df8m3.32.
All done
```

This page intentionally blank.

APPENDIX 3. Generating Good Partitions with DynaPart

Decisions made during the development of a DYNA3D/ParaDyn model can affect the scalability and efficiency of the simulation. The following are some insights to give the analyst an idea of the scalability of a ParaDyn job, and some tips on how to improve that scalability.

3.1 DYNA3D Objects that Restrict Scalability

The following objects can restrict the scalability of DYNA3D models in some cases.

Symmetry planes with failure
Follower forces
Nodal constraints
Sliding interfaces of type 1, or 4.
Tie-breaking shell slidelines
Tied node sets with failure
Rigid body joints
Shell-solid interfaces
One-dimensional slidelines

The reason that these objects restrict scalability will be discussed in the next section.

3.2 Special Node and Element Sets

The partitioning of a DYNA3D/ParaDyn model is constrained by boundary conditions and other options contained in the model. These boundary conditions and options will force nodes and elements to be assigned to a single processor rather than being divided across more than one processor. Nodes that need to be kept together are assigned to Special Nodal Point (SNP) sets. Each of the DYNA3D objects mentioned in the previous section generates a SNP set.

Associated with each of the Special Nodal Point sets is a Special Element (SE) set. The SE set consists of all elements that contain either (1) a segment containing only nodes that are in the corresponding SNP set (default), or (2) one or more nodes in the corresponding SNP set (if keyword *nosegbased* is used).

Some SE sets can be subdivided to improve scalability, and others cannot. All SE sets from objects in the above list can NOT be subdivided. Therefore those objects, including sliding interfaces of the specified types should be kept small to promote good scalability, because they cannot be subdivided. Large sliding interfaces that are NOT in the above categories can be used, but they will be divided across processors if necessary. If they are divided, this will result in a small (but not major) effect on scalability.

After subdividing large SE sets where possible and necessary, all elements in the resulting SE sets must be assigned to a single processor. Non-overlapping (disjoint) SE sets can be assigned to different processors, but each disjoint SE set must reside entirely in one processor. This means that the relative size of the largest disjoint SE set limits the maximum number of processors that can produce an efficient parallel simulation. For example, if the entire job contains 200000 elements and the largest disjoint SE set contains 20000 elements, then the largest number of processors that the job can be well-partitioned for is ten. If the analyst partitions for more than ten processors, there will be no significant speedup over the ten-processor partition. Since one processor will contain the 20000 elements of the SE set, it will have more computation than other the processors, and the other processors will be forced to periodically wait for the busy processor to complete its calculations.

In most models, if there are large SE sets, they are due to sliding interfaces. To reduce the size of the largest SE set, reduce the size of the largest sliding interfaces.

3.3 Statistical results in the DynaPart Log file

The statistics for an entire model and for the largest SE sets can be obtained from the log of a DynaPart run. There are several programs that are run by the DynaPart partitioning script, and these programs give job statistics during the partitioning operation. The following statistics for the entire job can be obtained from the screen output from the DynaPart module SNPGEN:

```
          Number of nodal points:                      157331
          Number of 8-node hexagonal elements:         142369
          Number of 2-node beam elements:                9646
          Number of 4-node shell elements:                  0
          Number of 8-node thick-shell elements:            0

 There were:
       10 Special Nodal Point sets.  The longest set contained
     2208 nodal points, and all sets combined contained a total of
    12690 nodal points.
 There were:
```

```
        10 segment sets.  The longest set contained
      2046 segments, and all sets combined contained a total of
     13938 segments.
  There were:
        10 Special Element SAND sets.  The longest set contained
         0 elements, and all sets combined contained a total of
         0 elements.
```

If there are no SNP sets, SNPGEN will tell you, and the job should partition well for any number of processors. In most problems there are SNP sets, and their size will determine the optimal number of processors for an efficient partition.

If there are one or more SNP sets, then the statistics for the disjoint special element sets can be determined from the screen output of DynaPart program AETS. The size of the disjoint special element sets will determine how scalable the model will be. The smaller the largest set is, the more processors the model can be efficiently partitioned for.

```
There were:
        11 disjoint sets.  The longest set contained
      2024 elements, and all sets combined contained a total of
     15066 elements.
```

After the partitioning is completed, the above statistics for the disjoint sets can also be determined by looking at the end of the .dse file. If the name of your job is *bigmodel*, then type the following command at the UNIX prompt:

tail bigmodel.dse

Program REDUCEGRF generates a graph representing the model, and combines all the elements in a disjoint special element set into a single vertex in the graph. The weight of this vertex represents the computational load for that set. The weight of the largest vertex will determine how many processors the model can be partitioned for while maintaining a good load balance. REDUCEGRF generates output to the screen that tells the maximum number of processors this job can be partitioned for with a possibility of having a good load balance:

```
 Output graph statistics:
  Sum of vertex weights:                 133240912
  Largest vertex weight:                   2940960
    at vertex:                              136954
    which represents special element set:        5
  Max # partitions for good load balance:       45
```

In this case we partitioned for only 32 processors, so we should get a well load-balanced partition.

The *Computational Load Balance* statistics are obtained from PARSKMETIS:

```
Load balance:          1.010997      Edge cuts:              427586
```

A balance of 1.00 is a perfect load balance (each processor nominally has the same amount of computation to do at each time step). In this case we obtained a load balance of 1.01, which is a good load balance.

The *Communication Load* statistics are given by DynaPart module PFGEN:

```
                              Number of      Number of
                                Node         Adjacent
                            Communications   Processors

      Average per processor:      2069.9          8.2
       Max on any processor:      3740           19
    Total on all processors:     66238          264
```

At each time step, there are a total of 66238 node communications that must take place. The average number of node communications (per processor) is 2069.9, and the processor needing to do the most communication needs to communicate 3740 nodes.

The performance is best when the number of communications (both the maximum on any processor and the total on all processors) is minimized. These statistics are useful when comparing the efficiency of different partitions of a model.

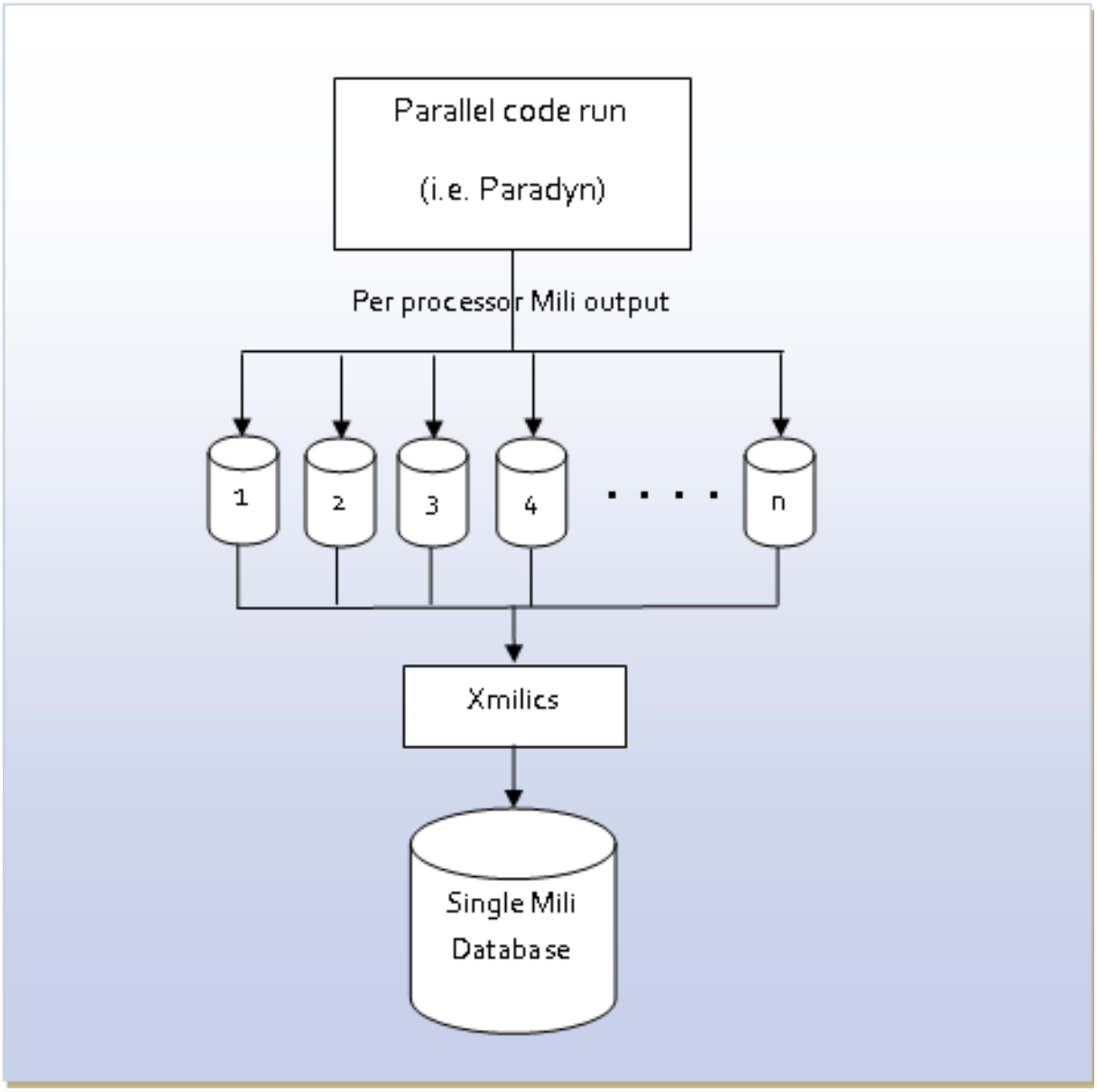
APPENDIX 4. XMilics User Guide

4.1 INTRODUCTION

Xmilics is a utility program developed by the Methods Development Group (MDG) to merge Mili database output created by parallel code runs. It thus consolidates ParaDyn visualization output into a form that the uniprocessor Griz visualizer can utilize.

4.2 XMILICS OVERVIEW

The following graphic provides a flow schematic for the role played by the xmiliCS utility in consolidating a set of visualization files suitable for use by the Griz post-processor.


Parallel code run
(i.e. Paradyn)
Per processor Mili output
1
2
3
4
n
Xmilics
Single Mili
Database

4.3 XMILICS OPTIONS

The following is a list of command line options that Xmilics supports.

Usage: xmilics -i <input_base_name> [-o <output_base_name>]

OPTIONS:

[-o] <output file name>
Create output file using this name.
Default output file name: {input_base_name}_c

[-1] <file size in megabytes>
Limit file size of combined database
Default file size: 10Mb

[-s] <number of states per file>
Limit number of states per file in
combined database
Default: none
NOTE: this option will override
the -l option

[-start] <state to begin at>
Limit the starting state of the Mili
database to be written states When
appending this is set to the last
time of output file if starting state
is less than the end state of the
output database.
Default start state: 1

[-stop] <state to end at>
Set the end state. This is the last
state to write to the new database. If

stop state is less than the last state of
the database to which we are appending then
this is ignored and the program exits with
nothing to do.
Default stop state: last state of input database

[-wait] <minutes to wait>

Wait for the files to be written. default is 10 minutes.

[-restart]

This tell Xmilics that it is awaiting a restart by paradyn. It
will wait the time specified by the -wait flag.

[-append] <state>

Append a timestep to an existing database. This is
overridden if a start or stop state are defined.

[-batch]

Run in batch mode. Disable interactive
and assumes append if output file exists
and -newfile is not set.

[-newfile]

Do not append - create new file

[-V] Display build stats (version-info)

4.4 XMILICS EXAMPLES

The examples use the following Mili databases:

Table 5

Full Database	Partial Database
Base Name: bar1.plt Processors: 8 Time Steps: 81 Partition File: bar1.dyn.8	Base Name: bar1_short.plt Processors: 8 Time Steps: 40 Partition File: bar1_short.dyn.8

A. Create a merged database from a full parallel run.
   a. Command: xmilics -i bar1.plt
   b. Output base name bar1.plt_c

B. Create a database from the full database but only up to the 10th time step.
   a. Command: xmilics -i bar1.plt -stop 10
   b. Output base name bar1.plt_c

C. Create a merged database from the partial database then append the additional information from the full database. Note that we use the -batch option to disable xmilics asking us if we want to append. Also we must run 2 separate xmilics runs.
   a. Command: xmilics -i bar1_short.plt -o bar1.plt_c
   b. Output base name bar1.plt_c
   c. Command: xmilics -i bar1.plt -batch
   d. Output base name bar1.plt_c

D. Create a merged database from the full parallel run starting at the 10th time step and ending at the 30th time step.
   a. Command: xmilics -i bar1.plt -start 10 -stop 30
   b. Output base name bar1.plt_c

E. Create a merged database from the full parallel up to the 10th time step then append the 20th and 30th time steps. Note that this requires multiple calls to xmilics at the moment.
   a. Command: xmilics -i bar1.plt -stop 10
   b. Output base name bar1.plt_c
   c. Command: xmilics -i bar1.plt -append 20 -batch
   d. Output base name bar1.plt_c
   e. Command: xmilics -i bar1.plt -append 30 -batch
   f. Output base name bar1.plt_c

F. Create a merged database from the full parallel using a partition file. Note that if time invariant

files are present you will receive a warning message that you will lose the time invariant data.

a. Command: xmilics -i bar1.plt -c bar1.dyn.8
b. Output base name bar1.plt_c

G. Create a merged database from the full parallel limiting 1 state per file.

a. Command: xmilics -i bar1.plt -s 1
b. Output base name bar1.plt_c

H. Create a merged database from the full parallel setting the file size to 100Mb

a. Command: xmilics -i bar1.plt -l 100
b. Output base name bar1.plt_c

I. Create a merged database from the full parallel limiting 1 state per file; only time steps 20 to 50.

a. Command: xmilics -i bar1.plt -s 1 -start 20 -stop 50
b. Output base name bar1.plt_c

This page intentionally blank

MANUAL CHANGE HISTORY

Version 11.1

September 2011

- Updates and incorporation of new Release Notes.

Version 10.1

September 2010

- Inclusion of Appendix 4 to fully document XmiliCS usage and incorporation of new Release Notes.

Version 9.1

September 2009

- Updates to command line syntax, removal of unsupported features, and incorporation of new Release Notes.

Version 8.1

September 2008

- Updates to command line sytax and incorporation of new Release Notes.

Version 7.1

December 2007

- Updates to command line sytax and incorporation of new Release Notes.

Version 6.1

September 2006

- Extensively rewrote the manual to upgrade it from the version published in February 2004.

- Documented how DynaPart keywords affect scalability when certain types of sliding interfaces are present. Reduced the number of types of sliding interfaces that can restrict the scalability of models when all keywords are defaulted.
- Updated Section 3.3 to illustrate the improved scalability of a model with slidelines that have local contact and tied contact in sliding interfaces.
- Updated Section 4.1 giving tips for designing models with efficient parallel contact.
- Revised Section 5 to be Released Notes rather than Future Enhancements.
- Updated Appendix 1 to more clearly specify when the DynaPart *again* keyword may and may not be used.
- Introduced new DynaPart keywords *badelemabort/nobadelemabort, smallgraph/nosmallgraph, tiedparallel/notiedparallel, and maxsettoconsider.* Eliminated keywords *agglom/noagglom*.
- Added several examples of DynaPart command lines.
- Updated Appendix 2 to report new documentation files and to show the current log file from a DynaPart run.
- Updated Appendix 3 to more clearly explain which DYNA3D objects restrict scalability, and why.
- Updated this Manual Change History.

Version 2.1/4.1

February 4, 2004

- Extensive rewrite of the manual to upgrade it from the manual published June, 2000.
- Provided information about the parallel visualization tools, VisIt and EnSight, throughout the manual.
- Discussions on parallel contact algorithms rewritten and updated in Section 2.4 on *Scalable Parallel Contact Algorithms*.
- Provided details on parallel automatic contact algorithms in Section 2.4.2.
- Added Section 2.4.3 and 4.1 on *Modeling Tips for Efficient Parallel Contact*.
- Added Section 2.6 on *Testing and Evaluating Model Scalability.*
- Revised and added paragraphs in Section 3.1 to include the parallel visualization tools.
- Added Section 3.2 to provide path variables for accessing the ParaDyn code and this documentation.
- Rewrote Section 3.3 on *Partitioning a Model.* Added examples illustrating partitioning models with and without contact.

- Updated Section 3.4 to add Mili database names and remove references to Taurus databases.
- Added Section 3.6 on *Running ParaDyn Interactively.* This provides example ParaDyn execution lines for several parallel computers using the utilities **mpirun**, **prun**, **poe**, and **srun**.
- Added Section 3.7 on *Running ParaDyn with Batch* with an example script and pointed to utilities for running under batch systems at the Livermore Computer Center.
- Updated Section 3.8 to focus on Mili databases (Section 3.8.1) and the three graphics post-processors:

  GRIZ in Section 3.8.2

  VisIt in Section 3.8.3

  EnSight in Section 3.8.4
- Updated Section 3.9 on *Steps for Running ParaDyn.* Provided a summary of steps in table form that can be printed as a reminder sheet.
- Deleted Section 4.1 on *Static Initialization and Dynamic Analysis***.**
- Added three Appendixes with detailed documentation for DynaPart.
- Added this Manual Change History.